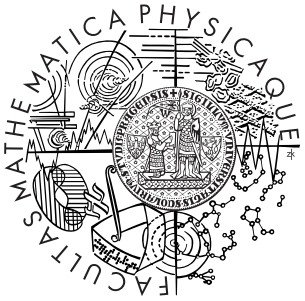
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Marie Konárová

Databáze studentů a jejich plateb

Katedra softwarového inženýrství

Vedoucí bakalářské práce: RNDr. David Hoksza
Studijní program: Obecná informatika

2009

Děkuji vedoucímu bakalářské práce RNDr. Davidu Hokszovi za poskytnutí cenných rad a literatury, potřebných pro zdárné ukončení práce. Dále děkuji četným přátelům a rodinným příslušníkům za pečlivé přečtení a jazykové korekce textu práce.

Prohlašuji, že jsem svou bakalářskou práci napsala samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce a jejím zveřejňováním.

V Praze dne Marie Konárová

Obsah

Název práce: Databáze studentů a jejich plateb
Autor: Marie Konárová
Katedra: Katedra softwarového inženýrství
Vedoucí bakalářské práce: RNDr. David Hoksza
e-mail vedoucího: David.Hoksza@mff.cuni.cz

Abstrakt: Předložená práce obsahuje návrh databáze studentů střední školy, mimoškolních aktivit organizovaných školou, docházky na tyto aktivity a plateb za ně. Její součástí jsou i funkce usnadňující tvorbu rozvrhu hodin. V práci je popsán ER model navržené databáze spolu s modelem datových toků (DFD). Součástí řešení je i definice uživatelských rolí a přiřazení práv těmto rolím. Databáze je samozřejmě chráněna proti nekonzistenci dat. Pro představu o možné podobě budoucí aplikace je implementována část uživatelského rozhraní. Ta zahrnuje většinu procesů spojených s jednou z uživatelských rolí. Kromě toho je implementován modul pro přihlašování uživatelů do systému. K práci je přiloženo CD se zdrojovými kódy pro vytvoření všech popsaných databázových struktur, triggerů a procedur. Kromě toho jsou přiloženy i zdrojové kódy vzorového uživatelského rozhraní.

Klíčová slova: databáze, rozvrhování, ER model, DFD diagram

Title: Database of students and their payments
Author: Marie Konárová
Department: Department of Software Engineering
Supervisor: RNDr. David Hoksza
Supervisor's e-mail address: David.Hoksza@mff.cuni.cz

Abstract: The present work contains a database scheme for managing data about high school students, school clubs or events, attendance to these activities and payments for them. It also includes functions to faciliate the scheduling. An entity-relationship model and a data flow model are described. The definition of user roles and their properties is also a part of the project. The database is protected against data inconsistence. A part of user interface is implemented to provide an example of the form of possible further application. The implemented part contains most of the processes supported by one of the user roles. Moreover, it contains a program unit for logging to system. The work includes a CD with source codes for creating all database structures, triggers and procedures. Source codes for the implemented part of user interface are also included.

Keywords: database, scheduling, ER model, data flow diagram

Kapitola 1

Úvod

Tato práce je motivována požadavky firmy Basis School, Inc., provozovatele sítě středních škol v USA, dále jako zadavatel. Úkolem práce bylo navrhnout komplexní systém pro administrativu základních a středních škol, který by nahradil nedostatky programu v současné době používaného zadavatelem[1]. V našem projektu se zabýváme především databázovou částí a jejím ošetřením. Soustředíme se též na robustnost a stabilitu dat za pomoci procedur a triggerů.

Databáze je pro školu přínosem při sledování stavu vypsaných akcí, přihlášek a docházek studentů na ně. Užitečnou funkcí navrženého systému je například evidence nové pohledávky vůči škole při vložení nové přihlášky studenta na vypsanou akci. To umožňuje lepší přehled o závazcích studentů vůči škole a nižší riziko opomenutí vymáhání platby. Databáze škole umožňuje hromadný pohled na shromažďovaná data a strukturované výstupy z jednotlivých tabulek či pohledů. Zpracovává osobní informace o učitelích, studentech a kontaktních osobách. Dále pak data o akcích, přihláškách, platbách a docházce na akce. Rovněž je možné v databázi evidovat majetek školy (budovy i vybavení místností). Součástí práce jsou i struktury usnadňující tvorbu rozvrhů.

Zadavatel požaduje, aby databáze pracovala se souhrnem událostí, které se dějí při mimoškolních činnostech, jako jsou přihlášky a docházka na sportovní kluby, či jiné zájmové kroužky. S tím souvisí i nutnost zpracování dat týkajících se příjmů školy od studentů za uvedené akce, kolekce soukromých

[1] Stávající aplikace například neobsahuje informace o studentech školy a jejich kontaktních osobách. Dále pak má nedostačující výstupy ohledně plateb (neumožňuje vypisování plateb podle jednotlivých studentů a platby za akce nelze zapisovat po splátkách).

dat o studentech (jejich kontaktní údaje), informací o učitelích (pozice, úvazek, kontaktní údaje) a kontaktních osobách (kontaktní údaje, vztah ke studentovi). Součástí databáze jsou i rozvrhy studentů, tříd (myslíme místnosti) a učitelů.

Do databáze vstupují tyto údaje:

- přihlášky na akce
- proběhlé platby
- základní data studentů, učitelů a kontaktních osob
- informace o předmětech (např. počet vyučovaných hodin týdně)

Z databáze vystupují následující údaje:

- přehledy plateb pro jednotlivé plátce
- docházka na příslušné akce
- bilance plateb z jednotlivých akcí
- rozvrhy hodin z pohledu tříd, žáků, učitelů a místností.

Výzvou v předkládané práci bylo řešení reprezentace rozvrhů a následné vypracování algoritmu generujícího rozvrh. Po nastudování několika používaných algoritmů jsme se rozhodli pro vytvoření vlastního algoritmu, který by byl dobře proveditelný v jazyce T-SQL. Pro tento algoritmus jsme navrhli příslušné struktury tak, aby bylo možné tvořit rozvrhy pro třídy žáků i pro každého ze studentů individuálně. Protože se však pohybujeme na střední nebo základní škole, jsou některé předměty pro všechny studenty jedné třídy či studijní skupiny stejné. Hlavní algoritmus na rozvrhování je ač jednoduchá, přesto účinná heuristika s malým nárokem na čas a rozumnými výsledky.

Administrační systém je doplněn o vzorovou implementaci uživatelského rozhraní a testovací data. Důležitou součástí vzorového uživatelského rozhraní je připravený systém přihlašování. Systém umožňuje přihlašování na dálku do aplikace běžící na serveru pod různými rolemi s odlišnými přístupovými právy. Narozdíl od stávající aplikace je tedy možné, aby byl systém využíván více lidmi zároveň, a jeho používání tak není vázáno na jediný

počítač. Vzorová implementace dále obsahuje návrh rozhraní pro práci s databází v rámci jedné z definovaných uživatelských rolí. Umožňuje přihlášení uživatele v této roli a používání aplikace s příslušnými právy.

Kapitola 2

Databázová část

Při návrhu databázové části práce byla v souladu s obecnými doporučeními, [2], prováděna normalizace. Ta pomáhá k odstranění redundantních dat, omezení složitosti (rozložením složité relace na dvojrozměrné tabulky) a zabránění tzv. aktualizačním anomáliím (např. abychom smazáním všech studentů třídy nepřišli o data o třídě). Normalizace by měla vést k databázi přehlednější, rozšiřitelnější a výkonnější.

Všechny relace navržené databáze splňují třetí normální formu (tedy splňují i podmínky pro první a druhou normální formu). Podmínkou pro splnění první normální formy je, že každý atribut obsahuje jen atomické hodnoty. Relace se nachází v druhé normální formě, jestliže je v první normální formě a každý neklíčový atribut je plně závislý na primárním klíči, a to na celém klíči a nejen na nějaké jeho podmnožině. Relace se nachází ve třetí normální formě, pokud se nachází a žádný z jejich atributů není tranzitivně závislý na klíči. Díky splnění této normy se databáze dále snadněji rozšiřuje.

Databáze zahrnuje osobní informace o studentech (např. jméno, bydliště, telefon), jejich kontaktních osobách včetně vztahů mezi studentem a jeho kontaktními osobami. Dalšími informacemi obsaženými v databázi jsou studijní plány (tj. navržené předměty s údaji o potřebné aprobaci na vyučování předmětu a týdenní hodinové dotaci), vypsané předměty (s informací o jim určeném vyučovacím čase, o učiteli a o místností, ve které se budou vyučovat) a osobní rozvrhy jednotlivých studentů. V databázi mohou být rovněž uloženy osobní informace učitelů včetně jejich úvazků, pozice, kterou zastávají na škole a jejich aprobací. Kromě toho jsou shromažďovány informace o budovách, které škola využívá či vlastní, o počtu místností v nich, jejich kapacitě a vybavení. Důležitými daty zpracovávanými databází jsou

informace o akcích vypsaných školou, přihláškách, docházkách a platbách za ně.

Navržená databáze je tedy velmi komplexní a umožňuje škole zpracovávat široké spektrum informací, které navíc mohou být logicky propojeny tak, jak to vyžaduje provozní praxe.

2.1 ER model

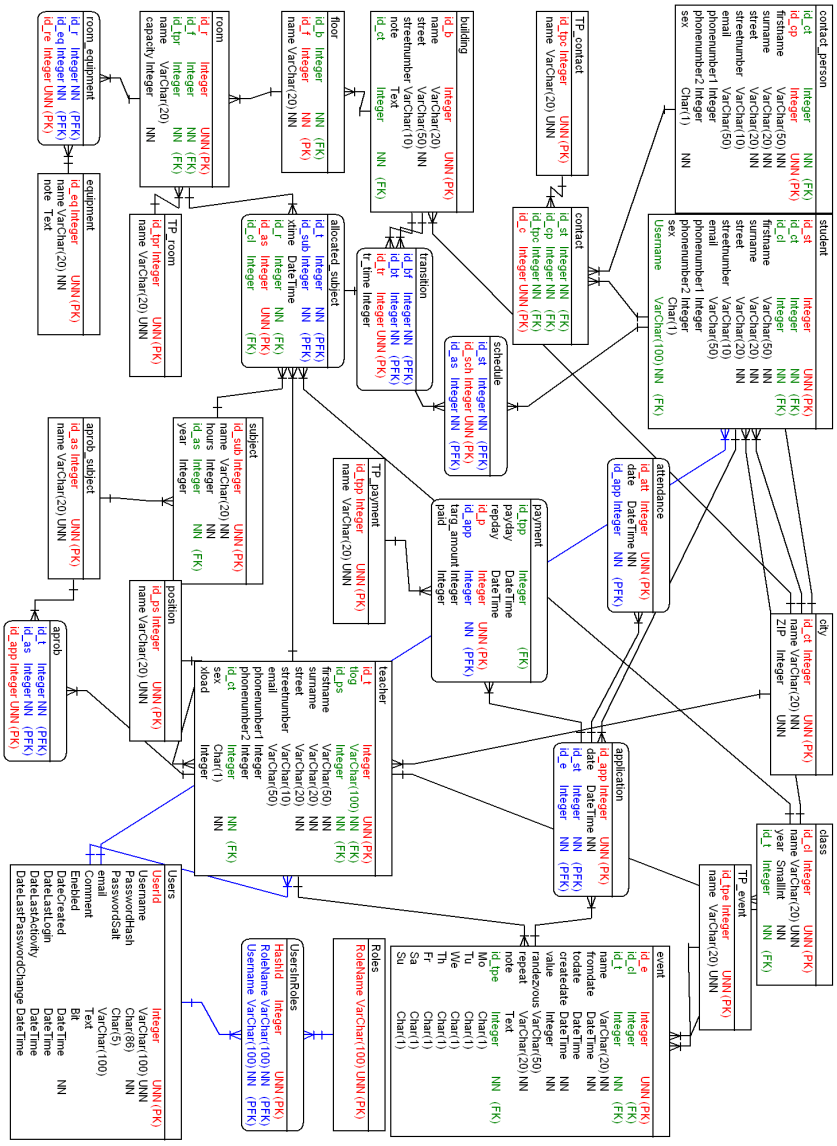
Obrázek 2.1 reprezentuje navržený ER model. Pro usnadnění orientace budeme využívat submodely, v nichž je méně entit a relace jsou zřetelnější. Model byl rozdělen na submodely rozvrhů, studentů, budov a učitelů. Některé entity jsou obsaženy ve více submodelech.

2.1.1 Submodel Student

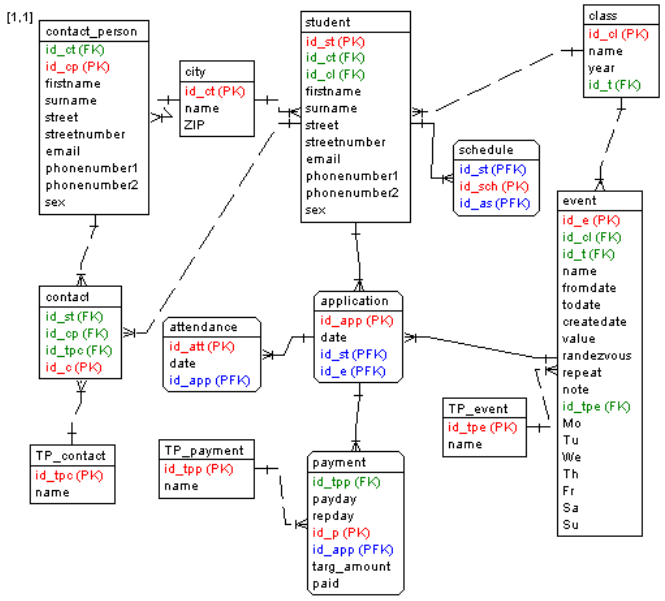
Submodel *Student* reprezentuje část databáze shromažďující informace o studentech, kontaktních osobách, akcích, docházkách a platbách studentů. Po založení záznamů studentů je možné upravovat jejich osobní údaje. Studentům lze přidávat kontaktní osoby pomocí entity *contact*, kde jsou zaneseny identifikační údaje studenta *id_s*, kontaktní osoby *id_cp* a typu kontaktu *id_tpc*. Informace o kontaktní osobě jsou reprezentovány samostatnou entitou (místo toho, aby byly součástí entity *student*), aby mohl každý student uvést více kontaktních osob. Za zmínku stojí entita *tp_contact*, která obsahuje informace o vztahu kontaktní osoby a studenta.

Dále je možné zakládat různé akce (entita *event*), které mohou být placené či neplacené, opakované či jednorázové (tyto a jiné možnosti jsou kvůli variabilitě zaznamenány v entitě *tp_event*). Akce mohou, ale nemusí být vázány na entitu *class*, což je skupina studentů. Na vypsané akce mohou být přihlašováni studenti v entitě *application* a pokud jsou studenti přihlášeni, může být sledována jejich docházka pomocí entity *attendance*.

Záznamy o platbách za akce se nacházejí v entitě *payment*. Platit je možné různými způsoby, které jsou zaneseny v entitě *TP_payment* (např. hotově, složenkou, převodem apod.). Platba za akci může být provedena ve více splátkách, kde každá splátka je reprezentována jedním záznamem v databázi. Sumy zaplacené za akci se pak sčítají přes všechny platby dané akce.



Obrázek 2.1: Navžený ER model



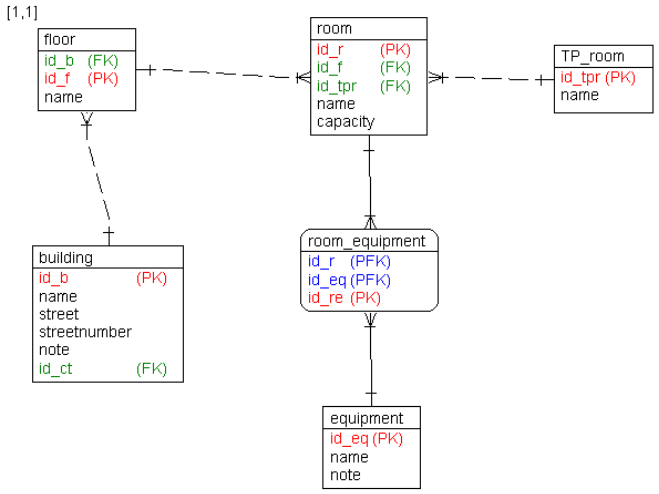
Obrázek 2.2: Submodel Student

2.1.2 Submodel Budovy

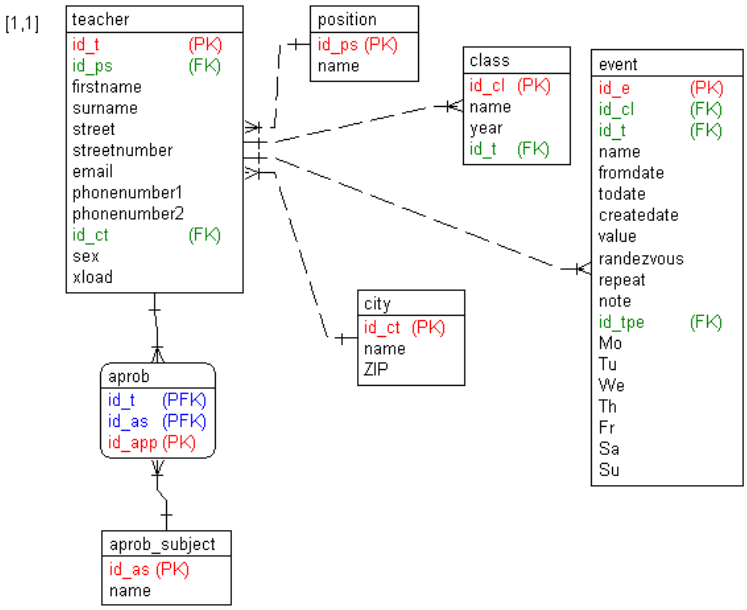
Submodel *Budovy* reprezentovaný obrázkem 2.3 slouží k evidování informací o majetku školy, jako jsou budovy a zařízení místností. V modelu je zahrnuta entita *building*, kde jsou informace o budovách. Protože potřebujeme vědět, jaké místnosti která budova obsahuje, máme v submodelu entitu *floor*, která obsahuje informace o patrech budov a entitu *room* obsahující informace o místnostech, jako je jejich typ a kapacita. V místnostech máme vybavení, které evidujeme pomocí entit *equipment* a *room_equipment*, kde entita *equipment* obsahuje informace o vybavení (např. počítače, televize, projektory, tabule) a entita *room_equipment* spojuje vybavení s jednotlivými místnostmi. Hierarchizace dat (budova, podlaží, místnost) usnadňuje změny v databázi např. při změně názvu ulice, ve které budova stojí.

2.1.3 Submodel Učitel

Submodel učitele, znázorněný na obrázku 2.4, slouží k evidenci záznamů o učitelích (entita *teacher*). Každý učitel musí mít danou pozici, kterou získá přiřazením záznamu patřícího do entity *position* (např. učitel, ředitel, zá-



Obrázek 2.3: Submodel Budovy

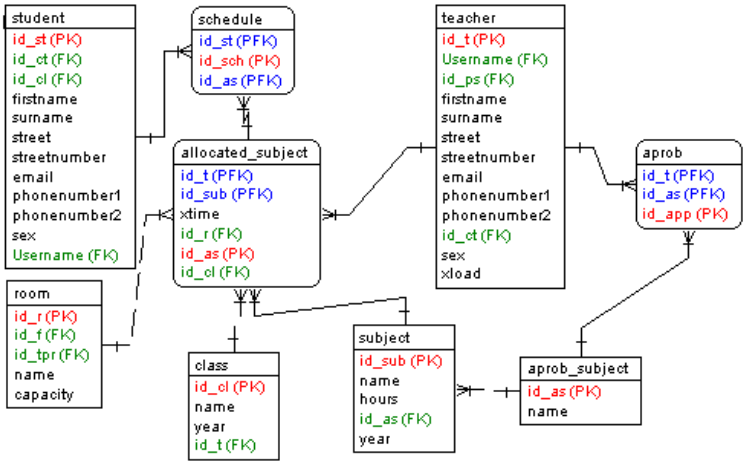


Obrázek 2.4: Submodel Učitel

stupce ředitele, správce učebny, atd.). Učitel může být třídním učitelem nějaké třídy, což se dozvíme v entitě *class*, pokud se tam nachází identifikační číslo daného učitele *id_t*. Na učitele může být vázána i akce (entita *event*), pokud na ni dohlíží, nebo za ni zodpovídá.

Pro tvorbu rozvrhů je důležité evidovat u každého učitele aprobaci na určitý předmět nebo předměty, což zajišťují entity *aprob* a *aprob_subject*. V entitě *aprob_subject* jsou předměty jako třeba matematika, fyzika nebo zeměpis a v entitě *aprob* jsou identifikátory předmětu *id_as* a učitele *id_t*. Významným atributem entity *teacher* je atribut *xload*, jehož hodnota určuje úvazek učitele, tj. počet vyučovacích hodin, které má daný učitel odučit za týden. Tento atribut je používán v rozvrhovacích funkcích popsaných v kapitole 3.2.2.

2.1.4 Submodel Rozvrhy



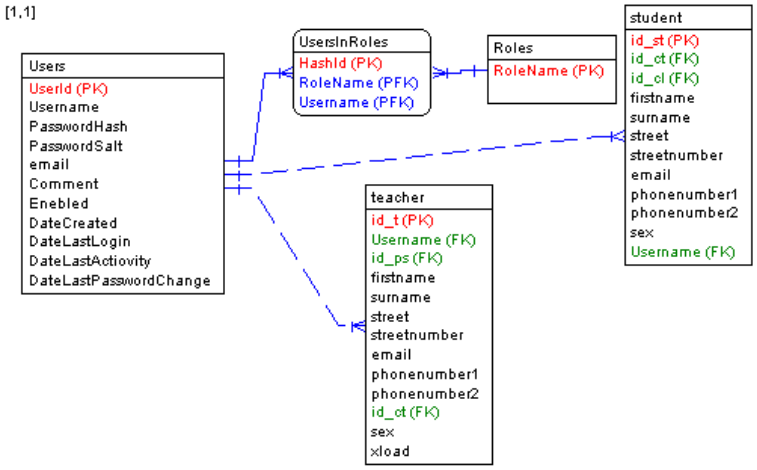
Obrázek 2.5: Submodel Rozvrhy

Předposlední ze submodelů, zobrazený na obrázku 2.5, tvoří entity a relace reprezentující rozvrhy. O entitách *teacher* a *aprob* jsme již hovořili výše (odstavec 2.1.3). Další entitou submodelu je entita *aprob_subject*, která reprezentuje předměty vyučované na škole nezávisle na ročníku nebo třídě. Tato entita je potřebná proto, že učitel má typicky aprobaci na konkrétní předmět bez ohledu na to, v jakém ročníku je vyučován.

S entitou *aprob_subject* je přes identifikátor *id_as* provázána entita *subject*, která rozlišuje předměty podle ročníků (atribut *year*) a zahrnuje i informaci o hodinové dotaci pro jednotlivé ročníky (atribut *hours*). Není-li v entitě *subject* uveden ročník, pak je předmět „meziročníkový“, tedy lze navštěvovat v libovolném ročníku. Tato entita je klíčová pro tvorbu rozvrhů. Obsahuje uživatelsky zadaná data, která jsou pak zpracována rozvrhovacím algoritmem.

Entita *allocated_subject* (rozvržené předměty) reprezentuje výstup procedury pro rozvrhování. Entita tedy obsahuje kromě ročníku i třídu a konkrétní čas, kdy je předmět vyučován (tj. např. matematika pro třídu A druhého ročníku v pondělí v 9:00). Pro každou hodinu v týdnu, kdy je předmět pro danou třídu daného ročníku vyučován, máme tudíž jeden záznam. Entita *schedule* umožňuje tvorbu individuálních rozvrhů pro jednotlivé studenty a to tak, že svazuje identifikátor studenta s identifikátorem vypsaného předmětu z entity *allocated_subject*.

2.1.5 Submodel Login



Obrázek 2.6: Submodel Login

Posledním submodelem našeho ER modelu je submodel určený k uchovávání loginů, rolí a jejich propojení se zbytkem databáze. Jeho zobrazení nalezneme na obrázku 2.6. Login by měl být propojen i se záznamem učitele

v databázi, pokud daný login patří učiteli se zmíněným záznamem. Obdobně to platí i u studenta. Tabulky *Users*, *Roles* a *UsersInRoles* byly vytvořeny podle požadavků knihovny *Altairis Simple ASP.NET SQL Providers*, kterou využíváme při správě uživatelských účtů a rolí.

2.2 Popis rolí

Z povahy projektu vyplývá, že není nutné mít dynamicky vytvářené role. Projekt tedy využívá pouze pevně dané role. Dle provedené analýzy požadavků nám bude stačit následujících šest rolí. Na role budou vázána práva přístupu k jednotlivým informacím.

První rolí je *administrátor*, který má v kompetenci správu loginů. Rozumíme tím jejich vytváření, přiřazování do rolí, navazování na vložené záznamy v databázi a jejich mazání.

Další rolí je *učitel*, který má právo prohlížet a měnit některé své záznamy, prohlížet informace o studentech, předmětech, vytvářet nové akce, vyplňovat docházku, či přihlašovat studenty na jím vypsané akce.

Uživatel v roli *účetního* má přístup k informacím o platbách studentů za jednotlivé akce. Má také možnost čtení bilancí studentů, či akcí.

Správce majetku má přístup k informacím o budovách, místnostech a jejich vybavení.

Předposlední role je určena pro *tvůrce rozvrhů*. Ten má přístup ke studijním plánům, místnostem a některým datům učitelů. Má k dispozici funkce databáze pro usnadnění tvorby rozvrhů.

Poslední rolí je *student*, který má přístup ke svým informacím, k vypsaným akcím a docházce na akce, k akcím na které je přihlášený. Může prohlížet vypsané předměty a svůj rozvrh. Rovněž má možnost přístupu ke čtení informací o svých proběhlých i neproběhlých platbách.

2.3 Diagramy datových toků

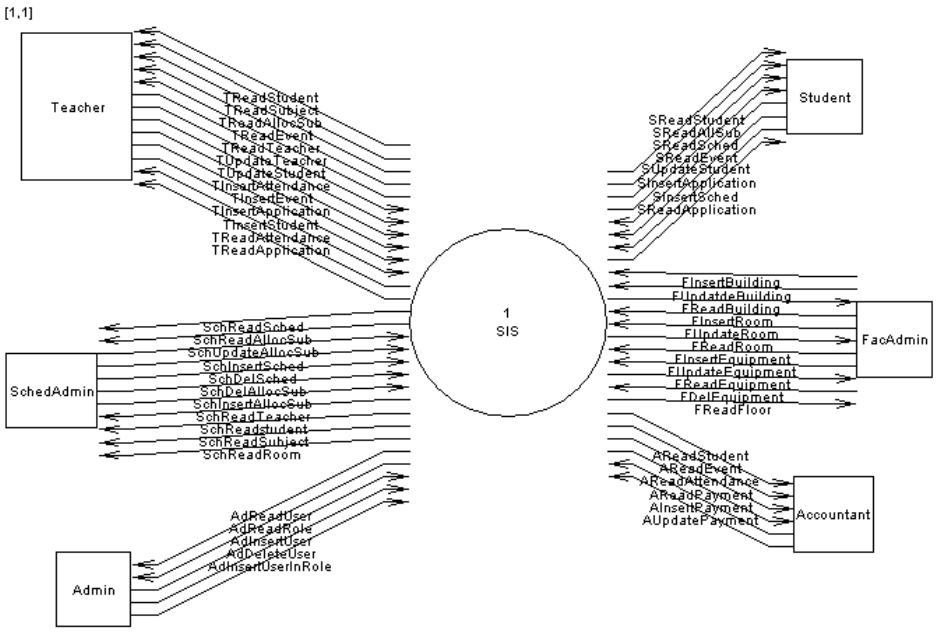
Diagramy datových toků (DFD) modelují procesy a toky dat mezi nimi. Lze z nich zjistit, jak jsou data užita a transformována při prostupu systémem. DFD diagramy jsou tvořeny čtyřmi stavebními prvky. Jsou to proces, tok dat, datastore a terminátor. DFD diagramy neobsahují řídicí informace a časování.

V prezentovaných diagramech se držíme následujících konvencí. *Proces* značíme zakulaceným objektem. Mezi jeho vlastnosti patří dekompozice (postupné zjednodušování). Dále musí mít alespoň jeden vstupní a výstupní datový tok a měl by být pojmenován. Procesy číslujeme hierarchicky tak, že například proces 1.1 nebo 1.2 je podprocesem procesu 1. *Tok* dat značíme čárou spojující proces se zbytkem systému, kde šipka označuje směr toku. Tok obvykle obsahuje i svůj název. Dále pak platí, že alespoň jeden konec musí být spojen s procesem a druhý konec musí být spojen s jiným procesem, datastorem, či terminátorem. *Terminátorem* rozumíme externí entitu, která je místem vzniku nebo příjmu dat systému. Značíme ho obdélníkem. *Datastore* je logický soubor dat, který obstarává nesynchronní přenos dat a skládání, či rozklad datových struktur. Musí obsahovat alespoň jeden vstupní a jeden výstupní tok. Značíme ho názvem mezi dvěma rovnoběžnými čarami. Značení bylo přebráno z [5].

Bereme v úvahu několik úhlů pohledu. Díváme se na model jako student, účetní, učitel, správce majetku, tvůrce rozvrhů a administrátor uživatelských účtů. Pro přehlednější zobrazení vytváříme víceúrovňové DFD diagramy. Diagramy ve vyšší úrovni jsou méně detailní (agregují detailnější DFD v nižších úrovních). Nejvýše v hierarchii stojí *kontextový diagram*, dále pak pokračujeme nultou úrovní, která je následována v tomto případě ještě dvěma dalšími úrovněmi. Procesy jsou číslovány. V nulté úrovni mají čísla 1.1, 1.2, atd., přičemž každému z terminátorů (rolí) přistupujících do systému odpovídá jeden proces nulté úrovně. V první úrovni se pokračuje v číslování tak, že například proces 1.4 je rozdělen do tří procesů, které jsou očíslovány 1.4.1, 1.4.2 a 1.4.3. Přitom platí, že každý proces první úrovně komunikuje s určitou skupinou logicky svázaných datastorů. Analogicky s číslováním pokračujeme i ve druhé úrovni, kde procesy odpovídají jednotlivým operacím s daty, například mazání, ukládání atd. Nejnižší úroveň by měla obsahovat minispecifikace zobrazených procesů. Procesy DFD diagramů nulté úrovně nemusejí mít stejný počet úrovní rozkladu, [5], čehož budeme využívat. Z důvodu přehlednosti DFD diagramů je dodržen doporučený maximální počet procesů v jednom diagramu šest až devět, [9].

2.3.1 Kontextový diagram

V kontextového diagramu (obrázek 2.7) jsou vymodelovány přistupující terminátory a jejich procesy. Každý z nich zastupuje jednu roli vstupující do aplikace, které jsou popsány v kapitole 2.2.



Obrázek 2.7: Kontextový diagram

2.3.2 Nultá úroveň DFD diagramů

Nultou, kořenovou, úroveň diagramu znázorňuje obrázek 2.8. Vidíme rozdělení systému do procesů, kde každý ze zobrazených procesů odpovídá jednomu z terminátorů (rolí) přistupujících do systému. Jeden proces v této úrovni obstarává veškeré požadavky přiřazeného terminátoru. Dále můžeme pozorovat některé datastory, jejichž počet je pro přehlednost omezen na ty, které považujeme za důležité pro pochopení práce systému.

2.3.3 První úroveň DFD diagramů

Z DFD diagramů první úrovně je vidět propojení procesů a datastorů. Procesy, které jsou rozvíjeny do další úrovně, budou podrobněji popsány v kapitole 2.3.4.

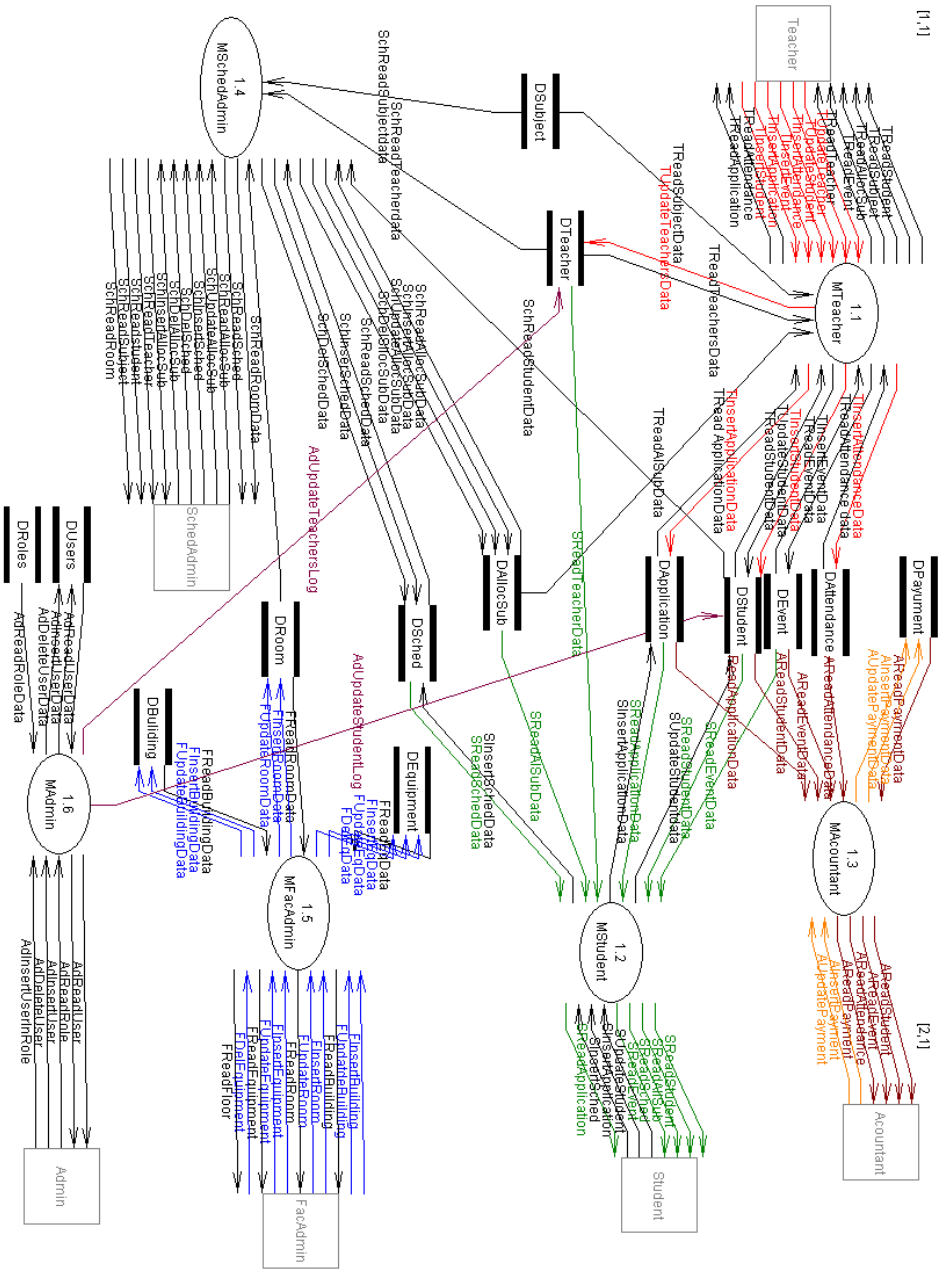
Na obrázku 2.9 vidíme DFD diagram první úrovně pro proces 1.1 *MTeacher* zpracovávající požadavky teminátoru *Teacher*. Proces 1.1.6 *TSubject* slouží k načítání dat o existujících předmětech. Podobně i proces 1.1.5 *TAllocSubject* slouží k načítání předmětů, ale jen těch, které jsou vypsány a mají přiřazeného učitele, případně i místnost a čas.

Obrázek 2.10 zobrazuje DFD diagram pro proces 1.2 *MStudent*, který zajišťuje komunikaci s terminátorem *Student*. Procesem 1.2.3 SAllocSubject zajišťuje čtení vypsaných předmětů. Proces 1.2.4. SEvent se rovněž zabývá čtením dat, a to jsou informace o akcích, tedy dat z datastoru DEvent, a přidružených informací jako je například zaštiťující učitel, či typ akce.

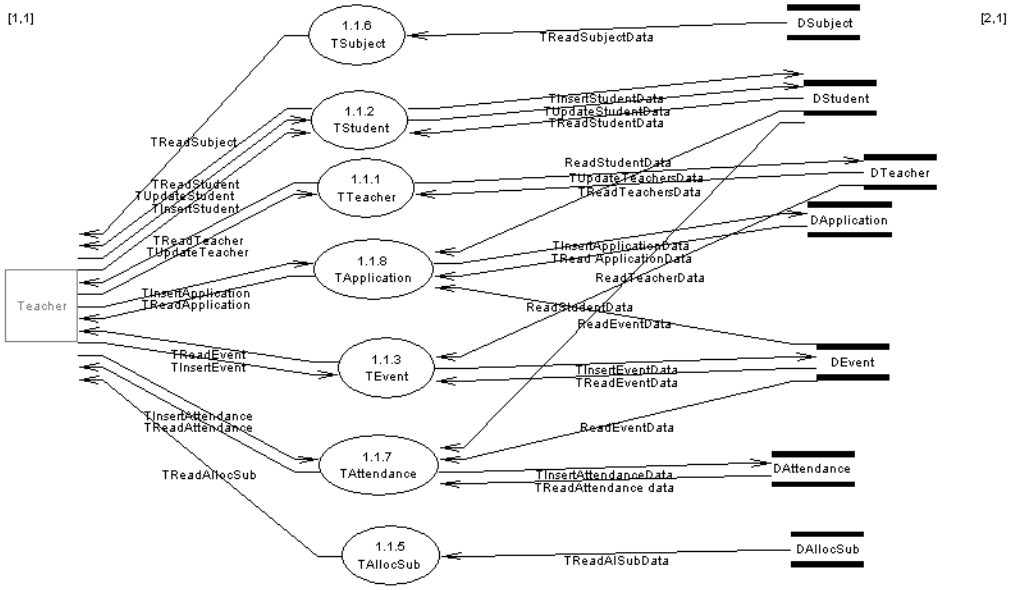
Procesy rozkládající proces 1.3 *MAccountant*, který zapracovává požadavky terminátoru *Accountant*, a končící rozklad v první úrovni, zobrazené na obrázku 2.11, zajišťují čtení dat. Proces 1.3.2 *AEvent* zařizuje čtení informací o akcích vypisovaných učiteli. Dále pak proces 1.3.3 *AStudent* čte dané roli přístupná data o studentech školy. Posledním procesem, který budeme specifikovat, je proces 1.3.4 *AAttendance*, který čte data o docházkách studentů.

V DFD diagramu zobrazeného obrázkem 2.12 a rozvíjející proces 1.4 *MSchedAdmin* (navázaný na terminátor *SchedAdmin*) se dále nerozkládá jediný proces a to 1.4.3 SchReadSubject, který obstarává čtení dat o existujících předmětech na škole terminátorem SchedAdmin.

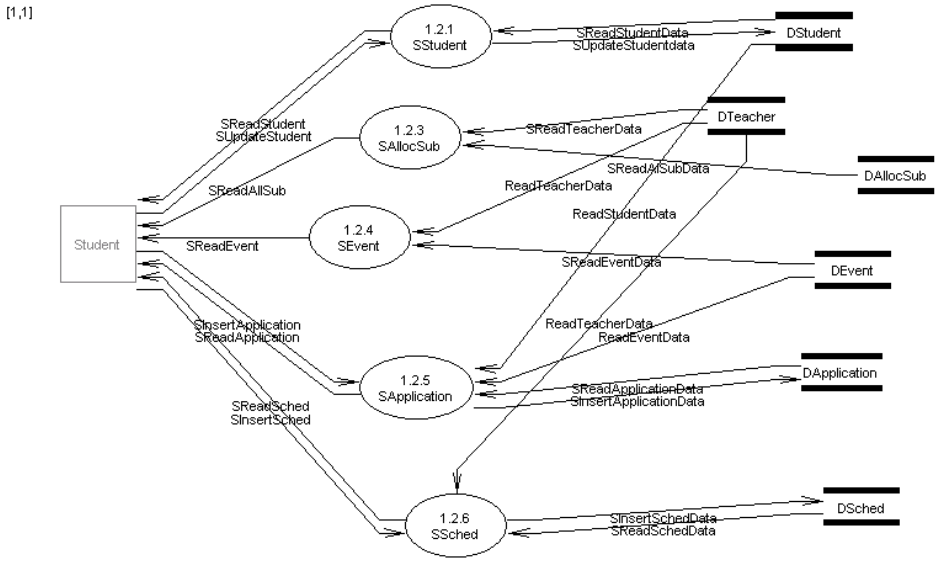
Předposlední diagram první úrovně je DFD diagram, který rozvíjí proces 1.5 *MFacAdmin* a je zobrazený na obrázku 2.13. V tomto diagramu dále rozkládáme všechny procesy do další úrovně.



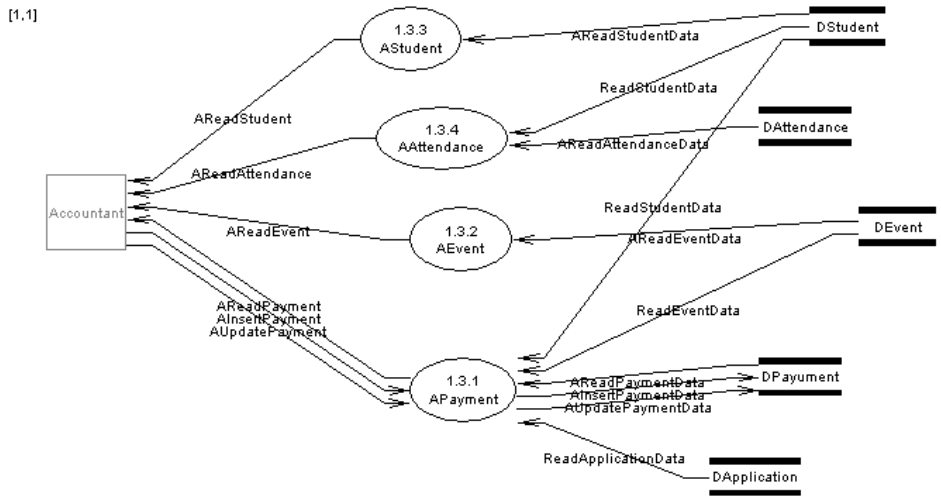
Obrázek 2.8: DFD nulté úrovně



Obrázek 2.9: DFD diagram první úrovně procesu 1.1 – MTeacher



Obrázek 2.10: DFD diagram první úrovně procesu 1.2 – MStudent



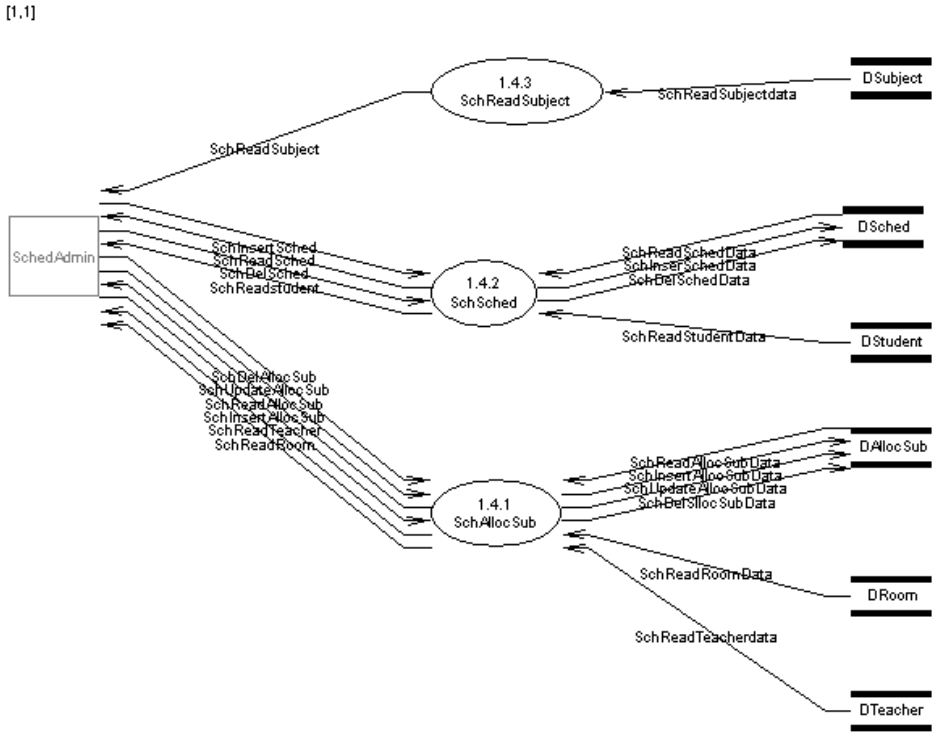
Obrázek 2.11: DFD diagram první úrovně procesu 1.3 – MAccountant

Posledním diagramem této úrovně je DFD diagram rozkládající proces 1.6 *MAdmin* komunikující s terminátorem Admin. Diagram je zobrazený na obrázku 2.14 a žádný z procesů tohoto diagramu není dále rozvíjen. Proces 1.6.1 *AdReadUser* čte data o uživatelích z datastoru *DUsers*. Proces 1.6.2 *AdDeleteUser* maže zvoleného uživatele z příslušného datastoru. Proces 1.6.3 *AdReadRole* čte záznamy o uživatelských rolích z datastoru *DRoles*. Proces 1.6.6 *AdInsertUserInRole* přiřazuje zvolené uživatele do vybraných rolí a vkládá záznamy do datastoru *DUserInRole*. Proces 1.6.4 *AdInsertUser* vkládá nového uživatele do datastoru *DUser*, následně předává informace o vloženém uživateli procesu 1.6.5 *AdUpdateLog*, který pak umožňuje napojení nového uživatele na záznam ve stávající databázi.

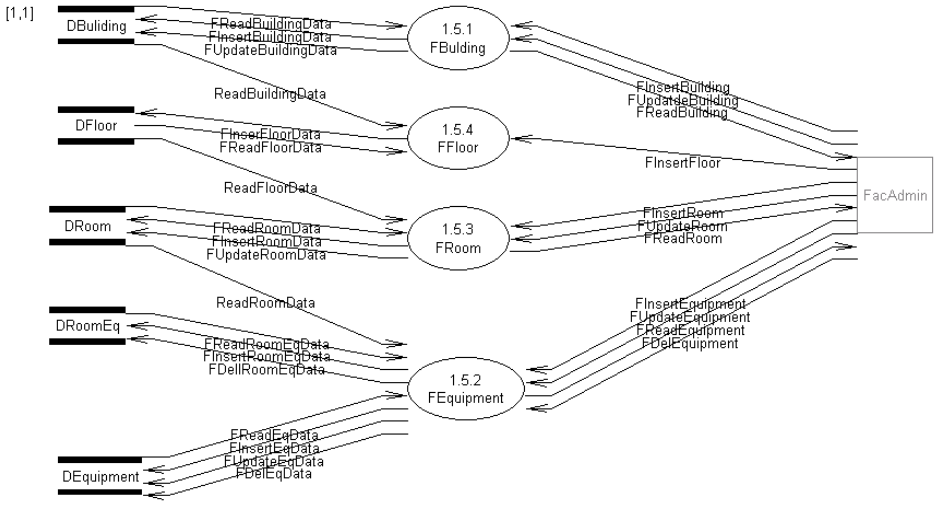
2.3.4 Druhá úroveň DFD diagramů

Druhá úroveň je poslední úrovní naší dekompozice. Bude tedy obsahovat minispecifikace všech procesů, které nebyly specifikovány v první úrovni. Protože jsou si některé obrázky velice podobné, jsou vybrané procesy popsány pouze slovně.

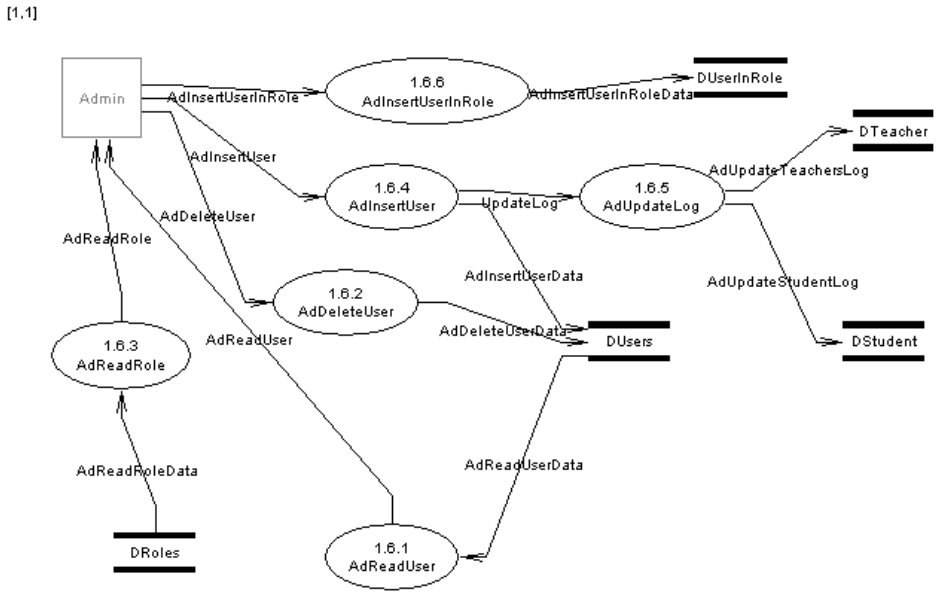
Proces 1.1.1 *TTeacher* je tvořen dvěma podprocesy. První z nich, proces 1.1.1.1 *TReadTeachersData*, slouží ke čtení dat o učiteli z datastorů *DTeacher*, *DCity*, *DAprob*, *DPosition* a *DAprobSub* (tabulky *teacher*, *city*, *aprob*,



Obrázek 2.12: DFD diagram první úrovně procesu 1.4 – MSchedAdmin



Obrázek 2.13: DFD diagram první úrovně procesu 1.5 – MFacAdmin



Obrázek 2.14: DFD diagram první úrovně procesu 1.6 – MAdmin

position a *aprob_subject*), kde jsou uloženy informace o učiteli, jeho aprobacích a městě, ve kterém má trvalé bydliště. Jsou čtena pouze data přihlášeného učitele. Dále pak proces 1.1.1.2 *TUpdateTeachersData* slouží i aktualizaci dat v datastoru *DTeacher*, která je možná přečíst v procesu 1.1.1.1.

Proces 1.1.2.1 *TReadStudent*, který je zobrazený na obrázku 2.15, zajišťuje čtení dat o studentech. Propojená skupina procesů 1.1.2.2 *TInsertStudent*, 1.1.2.4 *IStudent*, 1.1.2.5 *IContactP* a 1.1.2.6 *IContact* zajišťuje vkládání informací o studentech a jejich kontaktech. *TInsertStudent* přijímá informace o studentech a kontaktních osobách a předává informace procesům vkládající tato data do datastorů. *Istudent* vkládá data o studentovi do databáze, případně zjišťuje existenci studenta a dalšímu procesu předává identifikátor nalezeného nebo vloženého studenta. *IContactP* dělá totéž co *IStudent* až na to, že se jedná o nakládání s daty kontaktní osoby, dále je s těmito daty předávána informace o typu kontaktu. Data z předchozích dvou procesů jsou přijímána procesem *IContact*, který vloží vytvořený kontakt do datastoru.

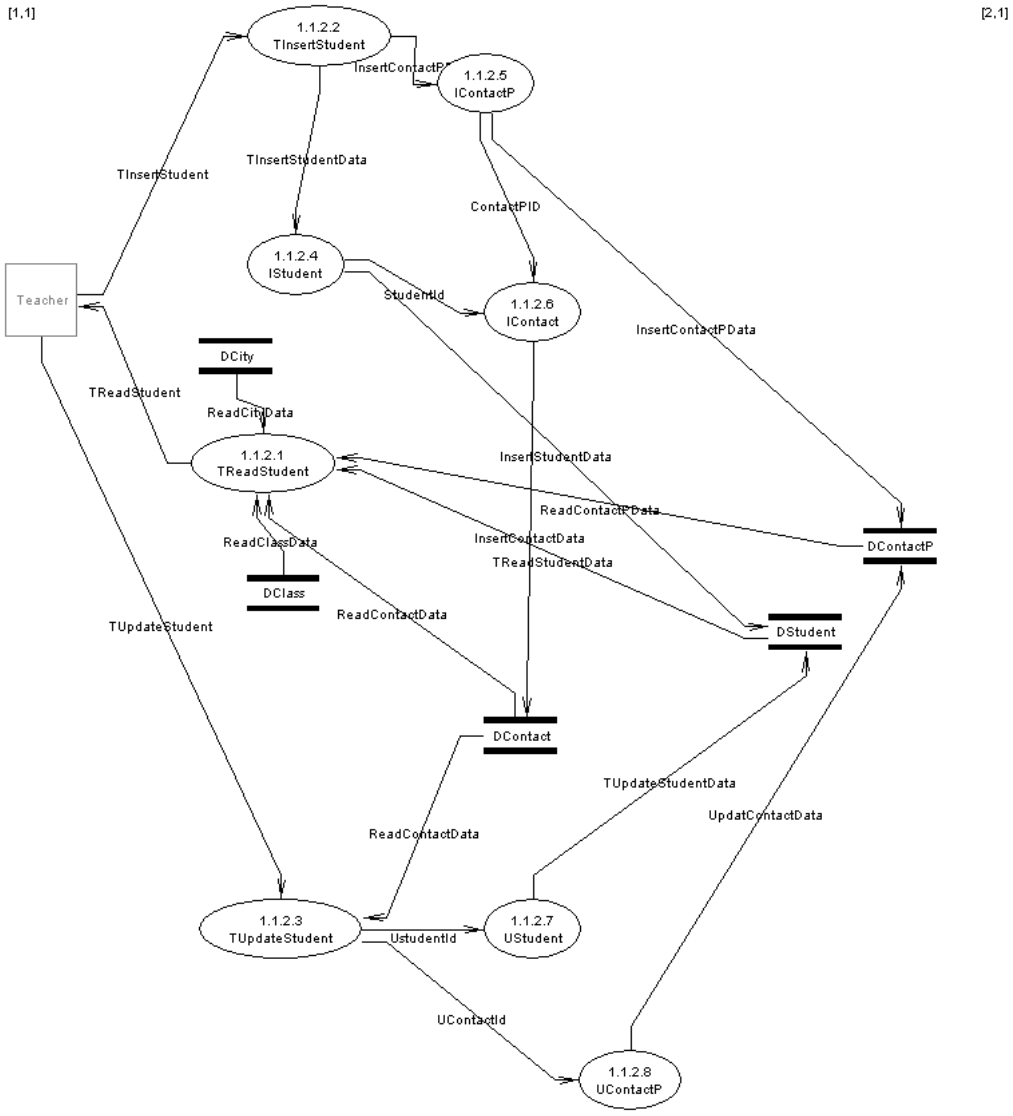
Na obrázku 2.16 vidíme DFD diagram druhé úrovně rozvíjející proces 1.1.3 *TEvent*. Prvním z procesů, které zde můžeme vidět je proces 1.1.3.1 *TReadEvent* zajišťující čtení dat o akcích a přidružených informací. Proces 1.1.3.2 *TInsertEvent* zajišťuje vkládání těchto informacích do příslušných datastorů.

Procesy rozkládající proces 1.1.7 *TAttendance* jsou pouze dva. Prvním je proces 1.1.7.1 *TInsertAttendance*, který zajišťuje vkládání docházky žáků na akce do datastoru *DAttendance*. Proces 1.1.7.2 *TReadAttendance* čte informace z datatastorů *DEvent* a *DStudent*.

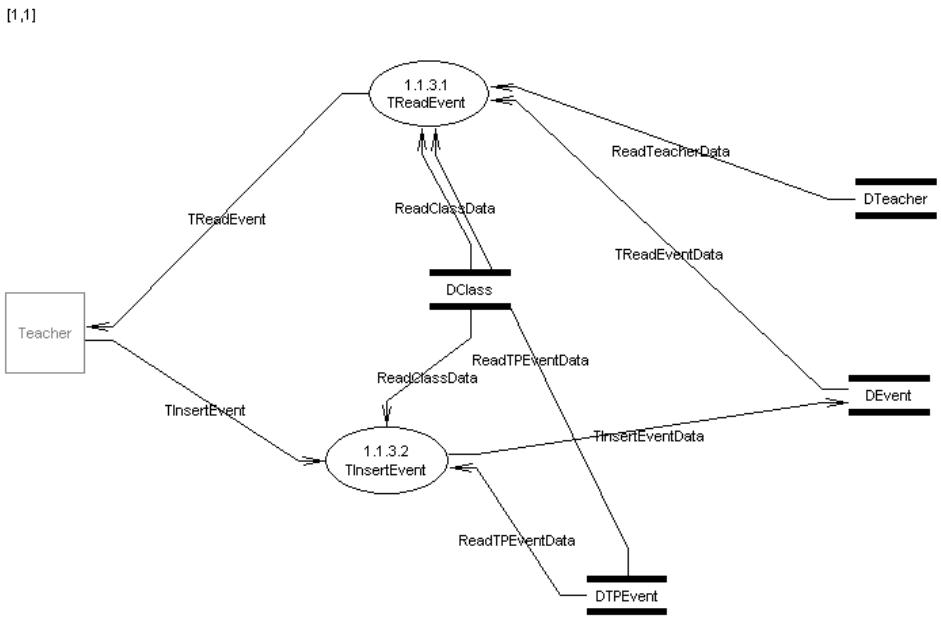
Na obrázku 2.17 opět vidíme pouze dva procesy, kde 1.1.8.1 *TReadApplication* obstarává čtení přihlášek na akce, tedy data z datastorů *DApplication*, *DStudent* a *DEvent* a proces *TInsertApplication* zajišťuje vkládání přihlášek do systému.

Obrázek 2.18 ukazuje diagram rozkládající proces 1.2.1 *SStudent*. Proces 1.2.1.1 *SReadStudent* zprostředkovává čtení dat o studentovi. Proces 1.2.1.2 *SUpdateStudent* zajišťuje přijetí aktualizovaných dat o studentovi a ty předává dále procesům 1.2.1.3 *SUStudent* a 1.2.1.4 *SUContactP*, které pak aktualizují příslušná data v datastorech.

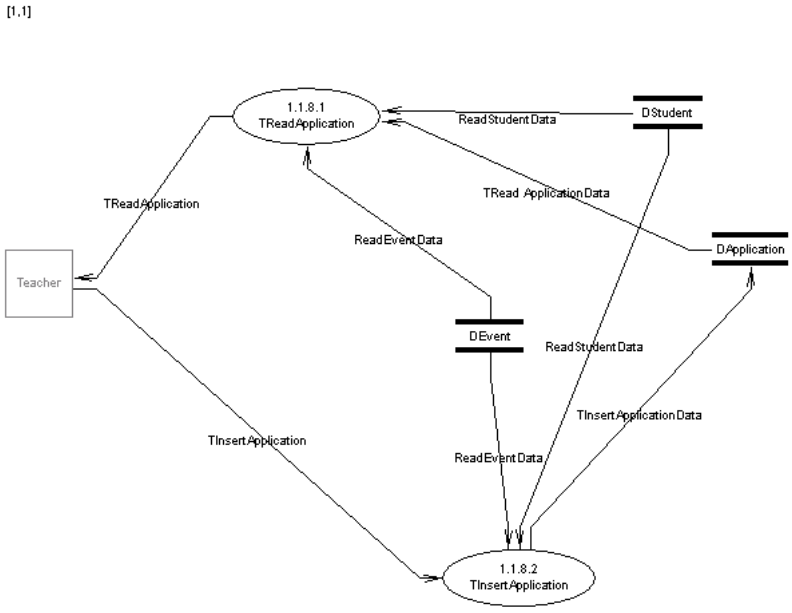
Procesy rozložené z procesu 1.2.5 slouží k přenosu dat o přihláškách. Proces 1.2.5.1 *SInsertApplication* slouží ke vkládání dat do datastoru *DApplication* s tím, že dostává identifikátor určeného studenta z datastoru *DStudent* (tabulka *student*) a identifikátor akce z datastoru *DEvent* (tabulka *event*) a proces 1.2.5.2 *SReadApplication* je používán ke čtení dat o přihláškách stu-



Obrázek 2.15: DFD diagram druhé úrovně procesu 1.1.2 – TStudent



Obrázek 2.16: DFD diagram druhé úrovně procesu 1.1.3 – TEvent



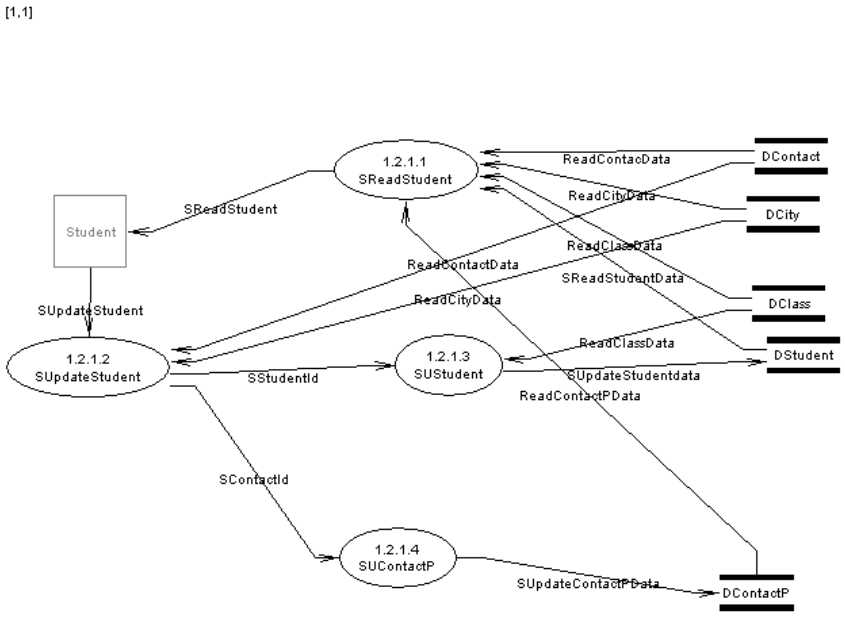
Obrázek 2.17: DFD diagram druhé úrovně procesu 1.1.8 – TApplication

denta přihlášeného na vybranou akci.

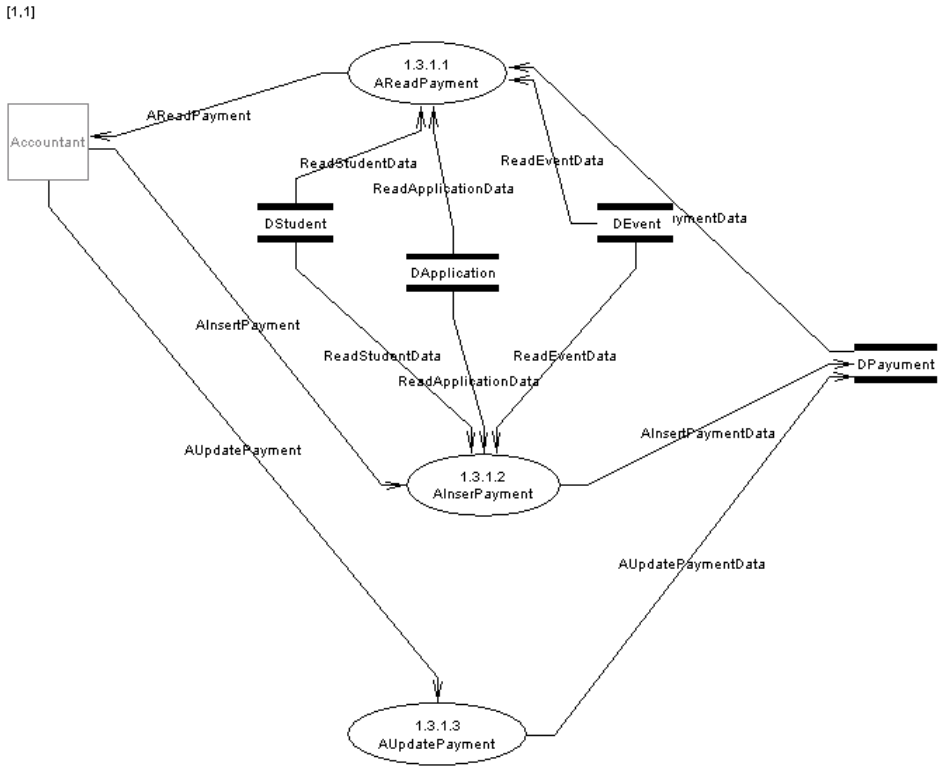
Posledním z diagramů sloužící ke komunikaci s terminátorem Student je diagram rozkládající proces 1.2.6 *SSched*. Obsahuje procesy 1.2.6.2 *SInsertSched* a 1.2.6.1 *AReadSched*. Oba slouží k manipulaci s daty o rozvrhu studenta a používají k tomu datastory *DSched* (tabulka *schedule*), *DTeacher* (tabulka *teacher*), *DStudent* (tabulka *student*) s tím, že poslední dva datastory slouží v obou případech pouze ke čtení.

Dále se podíváme na procesy spojené s čtením či přijímání dat rozvíjející proces 1.3.1 *APayment*. Všechny procesy z diagramu na obrázku 2.19 se týkají přenosu dat o platbách studentů. Začneme procesem 1.3.1.1 *AReadPayment*, který zprostředkovává čtení dat. Dále máme proces 1.3.1.2 *AInsertPayment*, který zprostředkovává vkládání dat do zobrazených datastorů. Posledním procesem je proces 1.3.1.3 *AUpdatePayment*, jehož prostřednictvím jsou data aktualizována.

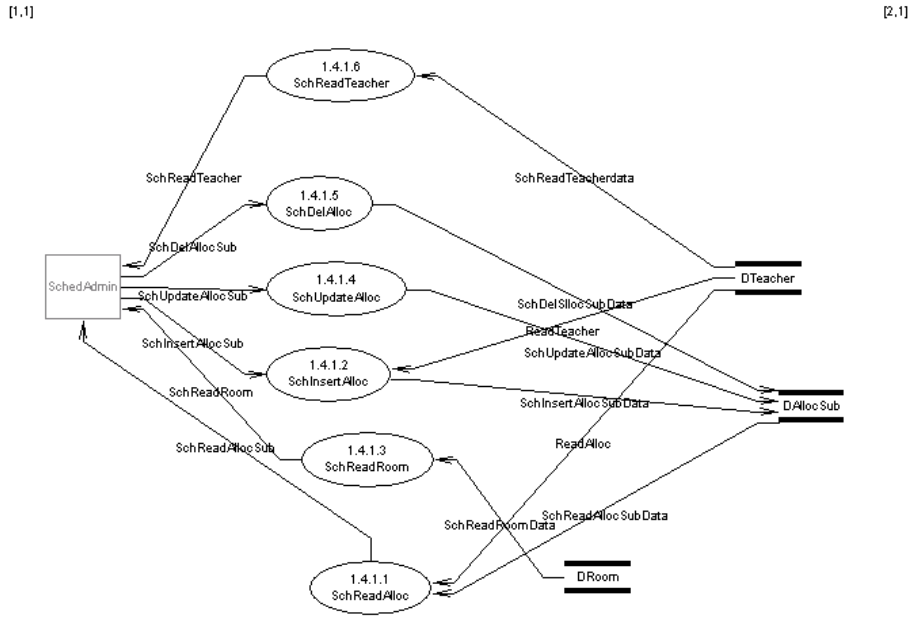
Na obrázku 2.20 vidíme zobrazené procesy, jejichž výstupy nebo vstupy jsou napojeny na terminál SchedAdmin. Většina procesů slouží ke čtení dat z příslušných datastorů jako třeba procesy 1.4.1.1 *SchReadAlloc*, 1.4.1.3



Obrázek 2.18: DFD diagram druhé úrovně procesu 1.2.1 – SStudent

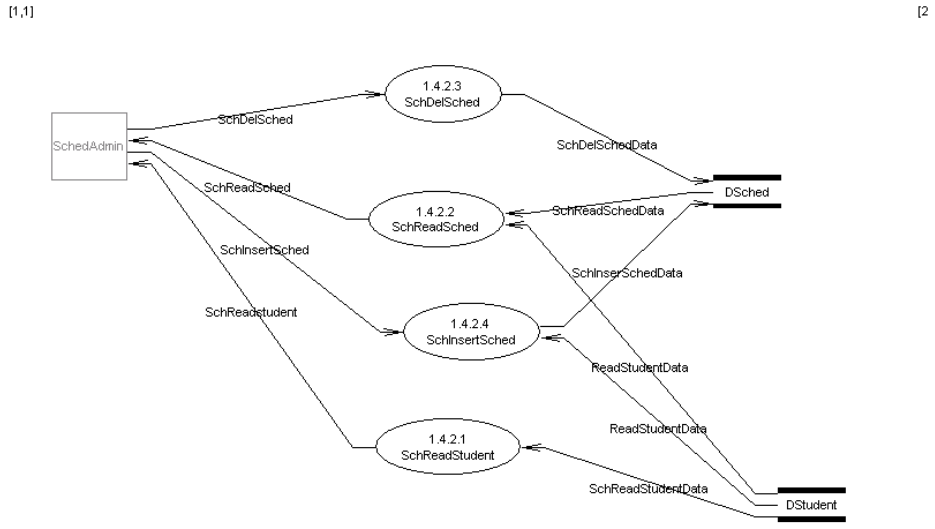


Obrázek 2.19: DFD diagram druhé úrovně procesu 1.3.1 – APayment



Obrázek 2.20: DFD diagram druhé úrovně procesu 1.4.1 – SchAllocSub

SchReadRoom a 1.4.1.6 *SchReadTeacher*. Proces 1.4.1.2 *SchInsertAlloc* slouží k ukládání informací o vypsaných předmětech. Proces 1.4.1.4 *SchUpdateAlloc* aktualizuje údaje o vypsaných předmětech a proces 1.4.1.5 SchDelAlloc slouží k mazání výše zmíněných předmětů.

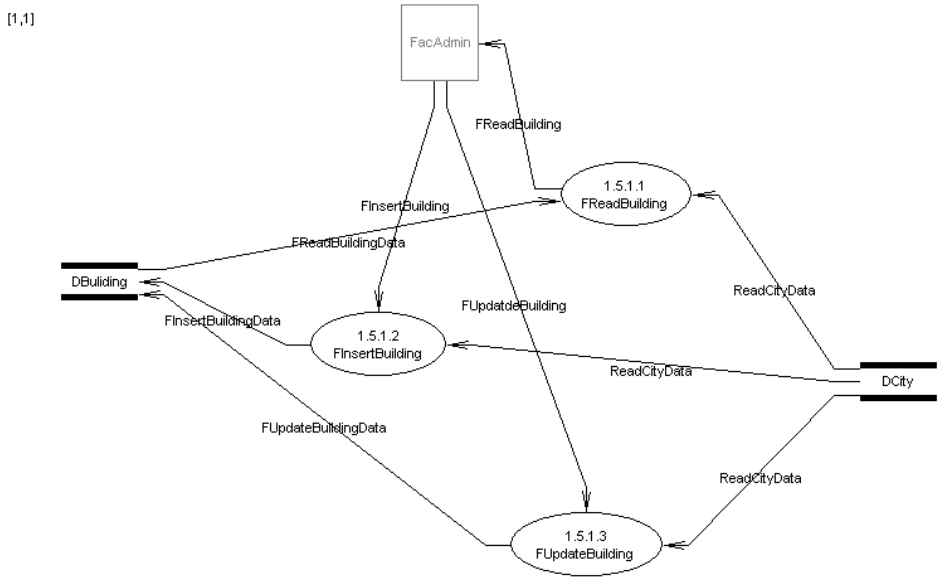


Obrázek 2.21: DFD diagram druhé úrovně procesu 1.4.2 – SchSched

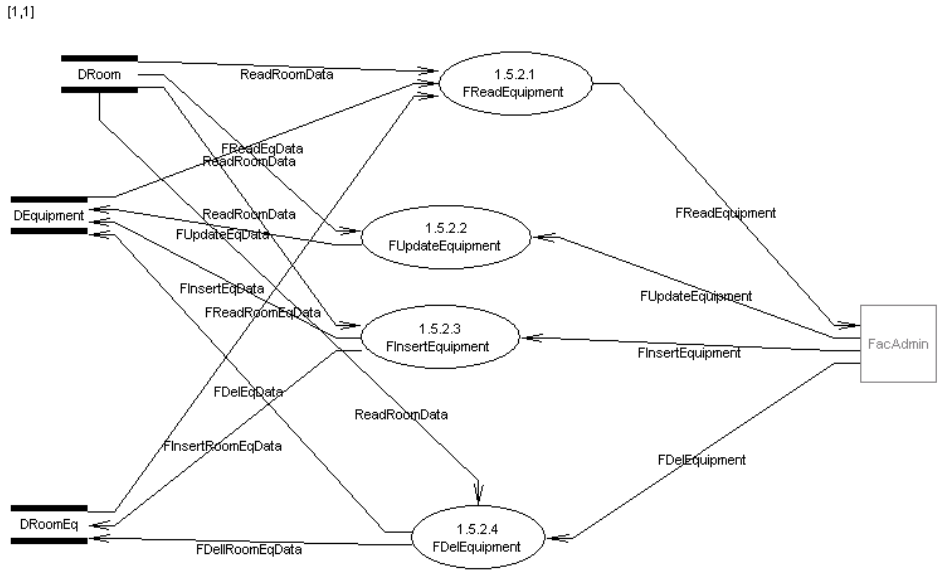
Druhá skupina procesů, která je napojena na terminál SchedAdmin, slouží k práci s daty souvisejícími s osobními rozvrhy žáků. Proces 1.4.2.1 *SchReadStudent* slouží ke čtení dat o studentech, proces 1.4.2.2 *SchReadSched* ke čtení rozvrhů, proces 1.4.2.3 *SchDelSched* k mazání rozvrhů a proces 1.4.2.4 *SchInsertSched* ke vkládání rozvrhů.

Na obrázku 2.22 vidíme procesy pracující s informacemi o budovách. Proces 1.5.1.1 *FReadBuilding* slouží ke čtení dat, proces 1.5.1.2 *FInsertBuilding* ke vkládání informací a proces 1.5.1.2 *FUpdateBuilding* k aktualizacím informací o budovách.

Druhý obrázek DFD diagramu zobrazující procesy spojené s terminátorem *FacAdmin* obsahuje procesy potřebné k práci s vybavením místností. Obsahuje proces 1.5.2.1 *FReadEquipment* čtoucí záznamy o vybavení místností, proces 1.5.2.2 *FUpdateEquipment* zajišťující aktualizaci údajů, proces 1.5.2.3 *FInsertEquipment* vkládající údaje o vybavení místností a proces 1.5.2.4 *FDelEquipmnet*, který zprostředkovává mazání výše zmiňovaných údajů.



Obrázek 2.22: DFD diagram druhé úrovně procesu 1.5.1 – FBuilding



Obrázek 2.23: DFD diagram druhé úrovně procesu 1.5.2 – FEquipment

Předposlední DFD diagram zobrazuje procesy pracující s informacemi o místnostech. Proces 1.5.3.1 *FReadRoom* čte informace o místnostech a o tom, v jakém je místnost podlaží (používá datastory *DRoom*, *DFloor* a tedy tabulky *floor* a *room*). Proces 1.5.3.2 *FInsertRoom* vkládá informace o místnostech do datastoru DRoom a přitom používá identifikátor podlaží z datastoru *DFloor*. Proces 1.5.3.3 *FUpdate* informace o pokojích aktualizuje.

Zbývající procesy rozložené z procesu 1.5.4 *FFloor* jsou procesy určené pro správu podlaží. Proces 1.5.4.1 *FInsertFloor* vkládá podlaží do datastoru *DFloor* napojeného na tabulku *floor* za současného přebírání identifikátoru z datastoru *DBuilding* napojeného na tabulku *building* a proces 1.5.4.2 *FReadFloorData* čte informace o podlažích a o tom, jaké podlaží je v které budově (opět datastory *DBuilding* a *DFloor*).

2.4 Procedury a triggery

2.4.1 Datová integrita

Datovou integritu chápejme jako označení pro přesnost, konzistenci a správnost dat v databázi. Máme definovaná pravidla, která tuto datovou integritu pomáhají zajistit. Pro popis dělíme tuto množinu pravidel do několika kategorií.

První kategorii nazveme sloupcovou integritou. Sloupcová integrita je určena pravidly platnými pro daný sloupec a lze ji definovat pomocí typu (např. sloupec *firstname* v tabulce *Student* má typ *VarChar*, což znamená, že bude obsahovat znaky a bude proměnné délky podle toho, jak dlouhé slovo v něm bude uloženo), délky dat (*VarChar* v předchozím příkladě má stanovenou délku 50 a tudíž se do databáze neuloží nic delšího než 50 znaků), povolení prázdné hodnoty atributu (použito např. u sloupce *year* v tabulce *subjekt*), rozsahu přípustných hodnot (použito v tabulce *class* u atributu *year*) a výchozí hodnoty (např. tabulka *attendance* atribut *date*). Další kategorii nazveme řádkovou integritou. Tato integrita vyžaduje, aby byly řádky každé tabulky jednoznačně identifikovatelné (Zaručujeme to autoinkrementálními primárními klíči v každé tabulce. Při vložení nového záznamu se hodnota klíče automaticky zvýší o jedničku, řádek je tam jednoznačně identifikovatelný pomocí tohoto klíče.). Předposlední kategorií je tzv. referenční integrita, která zajišťuje zachování relací mezi tabulkami. Do poslední kategorie spadají další pravidla, která nejsou uvedena v předchozích kategoriích.

2.4.2 Triggery

V naší databázi máme dva druhy triggerů. Triggery spouštějící se místo vkládaní a triggery vkládající se místo mazání. Součástí těchto triggerů je kontrola vkládaných či mazaných záznamů a jejich případné vložení či smazání. Pokud při kontrole najdeme chyby, reagujeme chybovou hláškou na výstup a operace, se kterou byl trigger spuštěn, se nevykoná.

K zajímavým triggerům patří trigger spouštěný při mazání studenta (záznamu z tabulky *student*) *tr_student_del*, který zároveň kontroluje, zda jsou všechny platby vůči škole uhrazené. Pokud nejsou, studenta není možné smazat. Dalším triggerem, který stojí za zmínku je *tr_class_del*, který je spouštěn při mazání třídy (záznamu z tabulky *class*) a hlídá, jsou-li v mazané třídě nějací studenti. Třídu se studenty není možné smazat. Tedy pokusíme-li se o mazání záznamu v tabulce tříd obsahující studenty, vrátí nám databáze chybovou hlášku a třídu nesmaže.

2.4.3 Uzamykání transakcí

Transakcí rozumíme jednu nebo více databázových operací, které je nutné provést všechny jako jednu nedílnou jednotku. Pokud při jedné z operací v transakci dojde k chybě, je třeba zrušit všechny operace této transakce. Může být provedena jen celá transakce a nikoliv jen její část.

Každá transakce by měla splňovat čtyři vlastnosti (atomičnost, konzistenci, izolaci a trvanlivost). Atomičností rozumíme, že transakce vystupuje jako jedna jednotka při vykonávání příkazů. Je splněná celá, nebo žádná její část. S konzistencí chceme, aby byla data po provedení transakce (v případě úspěchu i neúspěchu) v konzistentním stavu (zachovává integritu dat). Izolace znamená, že úpravy prováděné transakcemi máme izolované od úprav, které provádí jiné souběžné transakce. Po vlastnosti trvanlivost chceme, aby změny vyvolané potvrzenou transakcí zůstaly uložené v databázi trvale.

Uzamykání transakcí řešíme v našem projektu uvnitř procedur a funkcí v jazyce T-SQL. Veškeré operace na databázi provádíme s výjimkou čtení, skrze uložené T-SQL procedury. Chráníme tím přístup do databáze a umožňujeme odlišné možnosti úprav databáze různými rolemi.

2.4.4 Zajímavé procedury

Užitečnou procedurou v databázi je *PayForStudent*, která se použije, pokud se student rozhodne vyrovnat veškeré své dluhy vůči škole. Pro každou

akci (tabulka *event*) procedura vytvoří novou platbu, ve které je v atributu zaplatil (*paid*) doplněk toho, co už student zaplatil za danou akci k hodnotě, kterou má student za akci zaplatit. Zajímavými procedurami jsou také funkce vytvářející rozvrh. Ty jsou podrobně popsány v kapitole 3.

2.4.5 Vkládání aktualizace a mazání dat

V procedurách na vkládání, aktualizaci a mazání uzamykáme transakce, aby byla zajištěna konzistence dat (viz kapitola 2.4.3).

Procedura *tch_InsertStudentContact* ošetřuje, aby nebylo možné vložit pomocí role *teacher* studenta bez kontaktní osoby. Za jednu transakci je považováno vložení studenta do tabulky *Student*, kontaktní osoby do tabulky *ContactPerson* a vložení identifikátorů vložených záznamů spolu s typem kontaktu do tabulky *Contact*. Není tedy možné, za použití této procedury, vložit studenta bez jeho kontaktu.

Procedury jsou určeny pro určitou roli. To umožňuje povolit, aby například učitel mohl aktualizovat jiné údaje než samotný student. Procedury povolují aktualizovat jen vybrané údaje (například nelze měnit identifikátor záznamu).

2.5 Pohledy

Pohledem rozumíme virtuální tabulku založenou na sadě výsledků příkazu SELECT, [7]. Pohled obsahuje řádky a sloupce jako skutečná tabulka. Pole v pohledu pocházejí z jedné, nebo více reálných tabulek či pohledů. Data pocházející z pohledu jsou reprezentována, jako by pocházela z jediné tabulky. Ve standardním pohledu se data pohledu neukládají do databáze.

V našem projektu používáme pohledy pro zpřehlednění kódu a omezení počtu chyb u častěji kladených složených dotazů. Dále pak využíváme možnosti použití agregovaných funkcí. Například funkci COUNT v pohledu *vw_class*, který poskytuje informaci o třídě, jejím třídním učiteli a počtu žáků ve třídě. Pohledy nám rovněž umožňují výstup jen některých řádků, či sloupců pro určitou roli. Využíváme také možnost přejmenovávání sloupců u jednotlivých pohledů.

Kapitola 3

Rozvrhy

3.1 Problém rozvrhování

Problémy rozvrhování patří mezi NP-těžké problémy, [1]. Lze je formulovat jako úlohy celočíselného programování, nebo řešit klasickými optimalizačními metodami. Vzhledem ke složitosti problému lze však takto řešit pouze úlohy omezeného rozsahu. Metody však mohou být vybaveny heuristikami, které naleznou přijatelné řešení v rozumném čase.

V praxi nemusíme nutně dosáhnout optimálního řešení. Může nám stačit i řešení o něco horší, které ale dostaneme v rychlejším čase. Tak tomu bude i v této práci, kdy nebudeme požadovat optimální rozvrh, ale bude nám stačit rozvrh v jistém smyslu správný. Za správný rozvrh považujeme takový, že jeden učitel či žák není v jeden čas na více místech. V jedné místnosti pak musí být vyučován nejvýše jeden vyučovaný předmět. Jedinou optimalizací, kterou provádíme, je priorita časů. Snažíme se obsazovat přednostně časy co nejblíže k začátku vyučování v pracovních dnech jednoho týdne.

3.2 Navržený algoritmus

3.2.1 Předpoklady

Pro správný chod našeho algoritmu předpokládáme dostatečný počet vyučujících s dostatečně velkými úvazky. Druhý předpoklad považujeme za splněný, jestliže pro každý předmět z tabulky studijních plánů *subject* platí, že součet úvazků aprobovaných učitelů je větší nebo roven součtu hodinových dotací daného předmětu ve všech třídách. Navíc každý učitel s aprobací da-

ného předmětu musí mít úvazek v rozsahu alespoň rovném nejvyšší hodinové dotaci daného předmětu. Dostatečný počet vyučujících je takový, aby každý vypsaný předmět mohl být vyučován v rámci nějakého úvazku a předmět pro jednu třídu nebyl dělen mezi více učitelů. Dále pak předpokládáme dostatečný počet a kapacitu místností, ve kterých budou předměty vyučovány. Posledním předpokladem je, že studijní plán pro jeden ročník nepřesáhne čtyřicet hodin týdně. Což považujeme za rozumný předpoklad vzhledem k tomu, že se jedná o rozvrhy pro střední školu, kde je obvyklý počet hodin kolem třiceti.

3.2.2 Popis algoritmu

Generování rozvrhu je dvoukrokové. Vstupními daty jsou studijní plány (tabulka *subjects*), učitelé (tabulka *teacher*) a jejich aprobace (tabulky *aprob* a *aprob_subject*), místnosti (tabulka *room*) a třídy (tabulka *class*).

V prvním kroku procedura *GenerSubject* z tabulky se studijními plány (tabulka *subject*) vytvoří tabulku s předměty, které budeme chtít vyučovat (tabulka *allocated_subject*), a přiřadí jim vyučujícího. Pro každou hodinu vyučování, kterou by se měl předmět vyučovat je vytvořen jeden záznam v tabulce. Je vhodné, aby všechny hodiny jednoho předmětu pro určitou třídu vyučoval jeden vyučující, čehož se budeme držet. Nyní se podíváme na pseudo kód popisované procedury.

```
procedure GenerSubject
BEGIN
  predmet = SELECT id, year FROM subject
  ORDER BY DESC #hodin tydne
  WHILE predmet
  BEGIN
    trida = SELECT id FROM class
                      WHERE year = predmet.year
    WHILE trida OR year = NULL
    BEGIN
      ucitel = SELECT id,uvazek(xload) FROM teacher
                WHERE teacher ma aprobaci na predmet
                AND uvazek- predmet.#hodin tydne >
                                      >#jiz vypsanych hodin
      IF NOT ucitel
        mame malo ucitelu
```

```
      ELSE
      BEGIN
        FOR i = 1 TO predmet.#pocet hodin tydne
          INSERT VALUE TO allocated_subject
      END
      IF year IS NULL
        year = 0
        (nebude pak NULL a zastavi se cyklus s tridami)
    END
  END
END
```

Generování probíhá následovně. Pokud je předmět naplánovaný pro určitý ročník (je-li uveden v tabulce *subject* atribut *year*), tak každé třídě tohoto ročníku (třídy získáme z tabulky *class*) přiřadíme vyučujícího a vkládáme předmět i s přiřazeným vyučujícím do tabulky *allocated_subject* tolikrát, kolik máme určený počet vyučovaných hodin předmětu týdně (atribut *hours* v tabulce *subject*) násobený počtem tříd v daném ročníku. Nemáme-li daný ročník, pak předmětu přiřadíme učitele a počet řádků jen podle počtu vyučovaných hodin týdně. Tímto zajistíme dostatečný počet záznamů v tabulce *allocated_subject*, kterou budeme dále upravovat v druhém kroku.

V druhém kroku (procedura *Schedules*) přiřazujeme předmětům místnost a čas podle výše zmíněné priority časů. Dále pak postupujeme do pozdějších hodin. Rozvrh děláme přes všechny předměty. Načteme předmět, zjistíme třídu a nalezneme nejlepší volný čas třídy. Poté zjistíme, má-li v daném čase čas přiřazený učitel. Pokud ho má, hledáme místnost s dostatečnou kapacitou, ve které by mohl být předmět vyučován. Nemá-li čas učitel, nebo není-li volná žádná vhodná místnost, posuneme čas, od kterého hledáme volný čas třídy a vše opakujeme, dokud se nám čas třídy, učitele a vhodné místnosti nesejde. Zde je důležitý předpoklad, že máme dostatek místností na vyučování naplánovaných předmětů. Nyní následuje pseudokód procedury *schedules*

```
procedure Schedules
BEGIN
  predmet = SELECT id_predmetu, id_ucitele, id_tridy
                FROM allocated_subject
  WHILE predmet
  BEGIN
```

```
    WHILE nemame vyhovujici cas
    BEGIN
      cas = volny cas tridy
      IF ma cas ucitel predmetu
      BEGIN
        mistnost = trida s volnou kapacitou s volnym casem
        IF mistnost
        BEGIN
          aktualizuj predmet.cas = cas
        END
      END
      posun casu na nejblizsi vyhovujici
    END
  END
END
```

3.2.3 Vlastnosti algoritmu

Algoritmus je při dodržení předpokladů konečný. Rozvrhy nemusí být nutně optimální, nicméně díky výše zmíněné prioritě časů jsou většinou rozumně rozvrženy. Za dodržení předpokladů z kapitoly 3.2.1 a předpokládáme-li, že máme alespoň tolik hodin vyučovaných předmětů jako je učitelů, tříd (žáků), nebo místností a n označíme počet hodin vyučovaných předmětů, má náš algoritmus v nejhorším případě složitost $O(n^3)$.

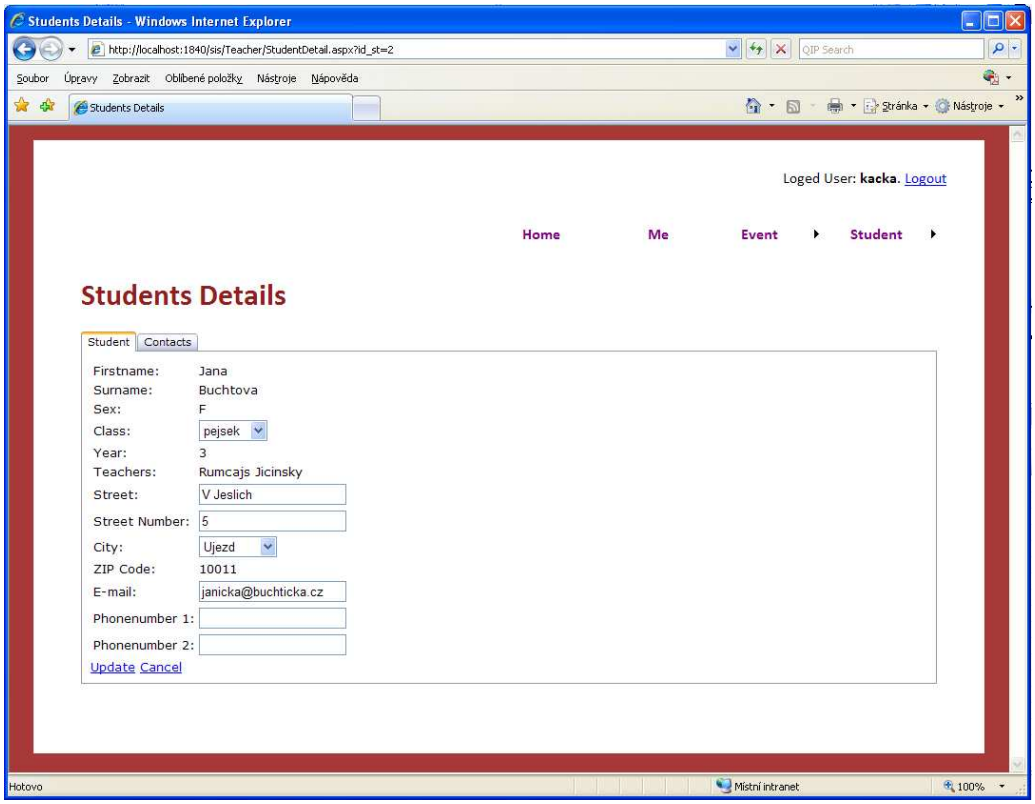
Kapitola 4

Vzorové uživatelské rozhraní

Vytvoření kompletního uživatelského rozhraní, dále jen UI, k projektu takovéhoto rozsahu je časově velice náročné a v praxi bývá zpravidla realizováno v dlouhodobé spolupráci se zadavatelem. Pro získání představy o možné budoucí podobě administračního systému však bylo implementováno vzorové uživatelské rozhraní. To obsahuje především modul pro přihlašování k aplikaci včetně přidělování uživatelských práv podle přiřazené uživatelské role.

Vzorové rozhraní obsahuje funkce pro role administrátora (vytváření a správa uživatelských účtů, přidělování rolí jednotlivým účtům a přiřazování účtů s rolí *student* nebo *teacher* k příslušným záznamům v databázi) a učitele. Přihlášený uživatel v roli učitele může upravovat svá osobní data, pokud jsou v databázi uložena, vkládat nové studenty a jejich kontaktní osoby, upravovat informace o existujících studentech (jak vidíme na obrázku 4.1) a o jejich kontaktních osobách. Navíc může vypisovat a prohlížet školní akce a upravovat data o akcích, které vypsal.

Nabídka dostupných funkcí se dynamicky mění v závislosti na roli přihlášeného uživatele. Jednotlivé položky nabídky jsou uloženy v souboru *web.sitemap*, z nějž se načítají pouze vybrané položky odpovídající příslušné roli. Při tvorbě vzorového uživatelského rozhraní jsme si rozšířili standardní prvky ASP.NET, [4] o AJAX extension, [8][10], která poskytuje několik nových prvků a velká rozšíření stávajících. V uživatelském rozhraní využíváme tzv. *master page*, což je jednotné pozadí zahrnující položky společné pro všechny obrazovky systému.



Obrázek 4.1: Obrazovka UI pro editaci údájů o studentech

4.1 Připojení k databázi

Pro připojení k databázi nám stačí jediný připojovací řetězec, který se nachází v souboru *web.config*. Ten můžeme snadno upravit pro potřeby námi používaného serveru. V souboru se mimo jiné nacházejí i údaje o právech. V jednotlivých stránkách pak již jen používáme tento nadefinovaný řetězec. Napojování jednotlivých prvků uživatelského rozhraní na příslušné databázové pohledy řešíme většinou standardními příkazy *bind* a *eval*. Rozdílně to například řešíme na stránce *EventDetail.aspx*, kde vypisujeme docházku studentů přihlášených na určenou akci. Zde jsou prvky tabulky pro akce tvořeny dynamicky.

4.2 Přihlašování

Na začátku tého kapitoly bych ráda zmínila možné způsoby autentizace v *ASP.NET*. Autentizace využívá čtyři různé módy, které nastavujeme v souboru *web.config* v sekci *Autentization*, [8]. V režimu *None*, který je nastaven jako výchozí, nedochází k žádné autentizaci. V režimu *Windows* ASP.NET autentizaci neovlivňuje. Zajišťuje ji operační systém MS Windows, který musí mít uživatelé nainstalován na počítači, z něhož se přihlašují. V režimu autentizace *Passport* předáváme řízení centrální službě MS Passport, která zajistí ověření identity uživatele. Údaje o uživateli jsou na centrálním serveru firmy Microsoft. Posledním režimem, který budeme používat v tomto projektu, je režim *Forms*. Používáme jej proto, že není vázán na operační systém přihlašovaného uživatele a variabilita tohoto režimu vyhovuje našim požadavkům pro přihlašování.

Součástí návrhu uživatelského rozhraní je i přidělení rolí uživatelům. Třídění dat, která mohou či nemohou vidět přihlášení uživatelé, je řešeno až v této vrstvě aplikace, kde povolením přístupu k určitým adresářům s jednotlivými stránkami odlišujeme od sebe přihlášené role. Ve stránkách pak rozlišujeme přihlášeného uživatele a zobrazujeme mu jen data, ke kterým má oprávnění přistupovat. Například přihlášený učitel může vidět a upravovat své osobní informace, ale nesmí upravovat informace o ostatních učitelích. Dalším příkladem je rozdílný pohled na vypsané akce, kde učitel vidí po přihlášení u jím vypsaných akcí více detailů a může k nim například zapisovat docházku přihlášeným studentům. Ke správě účtů používáme jedinou nestandardní knihovnu *ASP.NET* a to *Altairis Simple ASP.NET SQL Providers*, [6]. Práce v ní obsažených providerů s uživatelskými účty byla pro

řešení projektu výhodnější než práce standardních providerů.

Tvorbu nových loginů provádíme za pomoci prvku *CreateUserWizard*, který byl upraven, aby vyhovoval našim podmínkám. Jednou z úprav je možnost při přidání nového loginu vybrat, zda se bude vázat s nějakým záznamem databázi (buď studentem nebo učitelem), aby bylo možné zobrazovat data vázaná na přihlášeného uživatele. Navázání na stávající záznam není nutnou podmínkou vytvoření nového účtu.

Kapitola 5

SW požadavky

Provoz navrhovaného systému je možný pod operačním systémem Windows XP. Databázové skripty jsou napsané pro MS SQL 2005 EXPRESS, ale je možné je použít i pro plnou verzi tohoto SQL serveru. Pro uživatelské rozhraní je třeba mít nainstalovaný Misrosoft .NET Framework 3.5 a vyšší a dále požaduje ASP.NET server. Uživatelské rozhraní bylo programováno ve Visual Studiu 2005 za podpory jazyka VisualBasic.NET a pro testování byl používán ASP.NET Development Server.

Kapitola 6

Závěr

Naším úkolem bylo navrhnout systém pro administrativu základních a středních škol, který by nahradil a odstranil nedostatky současného systému zadavatele (Basis Schools Inc.). Kromě detailů o studentech umožňuje navržený systém evidovat údaje o zaměstnancích školy, vyučovaných předmětech a učebnách. Databáze umožňuje zpracování informací o školním rozvrhu, plánování mimoškolních aktivit, evidenci přihlášek, docházky a plateb spojených s těmito aktivitami. Databáze je plně funkční a kromě vlastních datových struktur zahrnuje prostředky k ochraně a kontrole konzistence dat, jako jsou například triggery, integritní omezení a procedury na uzamykání transakcí.

Součástí řešení jsou i procedury usnadňující tvorbu rozvrhu. Za stanovených předpokladů je možné automaticky generovat rozvrh hodin s preferencí dopoledních vyučovacích hodin a minimem prostojů mezi hodinami. Databázová struktura navrženého rozvrhu umožňuje pohled na rozvrh z různých úhlů dle uživatelských rolí. Navíc je možno navržený rozvrh ručně upravit podle aktuálních uživatelských požadavků.

Návrh databáze je přenositelný, nicméně praktická implementace je pro MS SQL server 2005 EXPRESS. Požadavek přenositelnosti byl jedním z důvodů pro opuštění myšlenky využití nástrojů MS SQL serveru pro správu uživatelských rolí a vedla k použití jiných struktur.

Implementace zahrnuje vzorové uživatelské rozhraní, které umožňuje uživatelský přístup k vybraným databázovým funkcím spojeným s rolí učitele. Kromě toho je implementován modul pro přihlašování uživatelů do systému, jehož součástí je i přiřazování uživatelských rolí, vytváření nových uživatelů a přiřazování práv dle role uživatele.

V zadání nebylo požadováno, aby systém kontroloval správnost zadaných studijních plánů nebo správnost zadávaných ZIP kódů vzhledem k městům. Nicméně o tyto a další detaily lze systém snadno rozšířit.

K práci je přiloženo CD se zdrojovými kódy v jazyce SQL pro vytvoření všech popsaných databázových struktur, triggerů a procedur. Dále pak soubor umožňující vložení testovacích dat do databáze. Je tedy možné vyzkoušet fungování databázové části systému. CD obsahuje i odkaz na běžící testovací verzi systému spolu s přihlašovacími údaji. Přiloženy jsou i zdrojové kódy vzorového uživatelského rozhraní v jazyce ASP.NET s využitím skriptovacího jazyka Visual Basic .NET [3].

Literatura

[1] Karp R.M: *Reducibility among combinotarial problems*. Complexity of Computer Computations 43, 1972, 95.

[2] Pokorný J., Halaška I.: *Databázové systémy: vybrané kapitoly a cvičení*, Karolinum, Praha, 1998.

[3] Roman S., Lomax P., Petrusha R.: *Visual Basic .NET v kostce*, Grada Publishing a.s., Praha, 2003.

[4] Rychter J.: *.NET Framework programování aplikací*, Grada Publishing a.s., Praha, 2003.

[5] Řepa V.: *Analýza a návrh informačních systémů*, Ekopress, Praha (1999) 403.

[6] Valášek M.: *Altairis Simple ASP.NET SQL Providers*, http://www.aspnet.cz/Articles/115-altairis-simple-asp-net-sql-providers-ke-stazeni.aspx, navštíveno dne 24.5.2009.

[7] Whalen E., Garcia M., Patel B., Misner S., Isakov V.: *Microsoft SQL Server 2005 Velký průvodce administrátora*, Computer Press a.s., Brno, 2008, 310 - 315.

[8] Woolston D.: *Pro Ajax and the .NET 2.0 Platform*, Apres, 2006.

[9] Yourdon E.: *Just Enough Structured Analysis*, Rev. 2006, 147 - 196.

[10] *ASP.NET AJAX Control Toolkit*,
http://www.asp.net/AJAX/AjaxControlToolkit/Samples,
navštíveno dne 24.5.2009.

Příloha A

Přehled použitých databázových struktur a procedur

A.1 Tabulky

student obsahuje osobní informace studentů.

class obsahuje informace o třídě žáků (např. ročník, třídní učitel).

attendance obsahuje informace o docházce studentů na akce.

payment obsahuje informace o platbách studentů.

TP_payment obsahuje možné typy plateb (hotově, kartou, atd.).

TP_event obsahuje typy akcí (taneční, hudební, sportovní, atd.).

contact_person obsahuje osobní informace o kontaktních osobách.

TP_contact obsahuje možné typy kontaktu studenta a kontaktní osoby.

contact spojuje studenta s kontaktní osobou.

city obsahuje informace o městě.

teacher obsahuje osobní informace o učitelích.

subject obsahuje informace o studijních plánech.

position obsahuje informace o možných pozicích zastávaných na škole.

aprob spojuje učitele s jeho aprobací.

room obsahuje informace o místnostech.

TP_room obsahuje možné typy místností (např. kabinet, učebna).

floor obsahuje název podlaží a informaci o budově ve které se nachází.

building obsahuje informace o budovách.

allocated_subject obsahuje rozvržené předměty.

application obsahuje informace o přihláškách studentů na akce.

aprob_subject obsahuje informace o možných aprobacích na předměty.

schedule obsahuje osobní rozvrhy studentů.

equipment obsahuje informace o zařízení používaném školou.

room_equipment přiřazuje zařízení školy do místností.

event obsahuje informace o vypsaných akcích.

transition obsahuje informaci o časových vzdálenostech mezi budovami.

users obsahuje uživatelské účty.

roles obsahuje uživatelské role.

UsersInRoles přiřazuje uživatelské role uživatelským účtům.

A.2 Triggery

tr_student_ins ošetřuje vkládání informací o studentech

tr_class_ins ošetřuje vkládání tříd

tr_attendance_ins ošetřuje vkládání docházky

tr_payment_ins ošetřuje vkládání plateb studentů za akce

tr_TPpayment_ins ošetřuje vkládání typů plateb

tr_TPevent_ins ošetřuje vkládání typů akcí (sportovní, hudební, atd.)

tr_contactperson_ins ošetřuje vkládání informací o kontaktních osobách

tr_TPcontact_ins ošetřuje vkládání typů kontaktů (právní, otec, atd.)

tr_contact_ins ošetřuje vkládání kontaktu

tr_city_ins ošetřuje vkládání nových měst

tr_teacher_ins ošetřuje vkládání učitelů

tr_subject_ins ošetřuje vkládání studijních plánů

tr_position_ins ošetřuje vkládání pozice

tr_aprob_ins ošetřuje připojování aprobací danému učiteli

tr_room_ins ošetřuje vkládání místností

tr_TProom_ins ošetřuje vkládání typů místností

tr_floor_ins ošetřuje vkládání podlaží

tr_building_ins ošetřuje vkládání budov

tr_allocatedsubject_ins ošetřuje vkládání vypsaných předmětů

tr_application_ins ošetřuje vkládání přihlášky (existence studenta, akce)

tr_aprobsubject_ins ošetřuje vkládání možných aprobací

tr_schedule_ins ošetřuje vkládání nových předmětů do rozvrhu studenta a hlídá zda vkládaný předmět nekoliduje s jiným již vloženým.

tr_equipment_ins ošetřuje vkládání zařízení do databáze

tr_roomequipment_ins ošetřuje připojení zařízení k určitému pokoji

tr_event_ins ošetřuje vkládání informací o akcích

tr_student_del ošetřuje mazání záznamu studenta

tr_class_del ošetřuje mazání záznamu třídy

tr_attendance_del ošetřuje mazání záznamu o docházce

tr_payment_del ošetřuje mazání záznamu o platbě

tr_TPpayment_del ošetřuje mazání záznamu o typu platby

tr_TPevent_del ošetřuje mazání záznamu o typu akce

tr_contactperson_del ošetřuje mazání záznamu kontaktní osoby

tr_TPcontact_del ošetřuje mazání typu kontaktu

tr_contact_del ošetřuje mazání záznamu s kontaktem

tr_city_del ošetřuje mazání záznamu města

tr_teacher_del ošetřuje mazání záznamu učitele

tr_subject_del ošetřuje mazání záznamu o předmětu ve studijním plánu

tr_position_del ošetřuje mazání záznamu o pozici

tr_aprob_del ošetřuje mazání záznamu o aprobaci učitele

tr_room_del ošetřuje mazání záznamu místnosti

tr_TProom_del ošetřuje mazání záznamu o typu místnosti

tr_floor_del ošetřuje mazání záznamu podlaží

tr_building_del ošetřuje mazání záznamu budovy

tr_allocatedsubject_del ošetřuje mazání záznamu rozvrženého předmětu

tr_application_del ošetřuje mazání záznamu o přihláškách studentů na akce

tr_aprobsubject_del ošetřuje mazání záznamu o aprobacích

tr_schedule_del ošetřuje mazání záznamu rozvrhu studenta

tr_equipment_del ošetřuje mazání záznamu o vybavení

tr_roomequipment_del ošetřuje mazání záznamu o vybavení v místnosti

tr_event_del ošetřuje mazání záznamu akcí

A.3 Procedury a funkce

TeacherHasTime pokud má vyučující ve vybraném čase volno, vrací true, jinak vrací false.

RoomHasTime vrací identifikátor třídy s volným vybraným časem a dostatečnou kapacitou pro niž je hledána.

ClassHasTime vrací první volný čas zvolené třídy .

HasAttendance vrací hodnotu true, pokud byla již na akci vložena docházka se zadaným čase. Jinak vrací false.

PayForStudent po spuštění tato procedura aktualizuje všechny nezaplacené platby zadáného studenta jako zaplacené.

GenerSubject vkládá záznamy s informacemi ze studijních plánů do rozvržených předmětů.

schedules aktualizuje informace o rozvržených předmětech a přiřazuje jim místnost a čas.

tch_UpdateTeacher používá se z uživatelského rozhraní na aktualizaci dat učitele. Obsahuje uzamknutí transakce.

tch_InsertStudentContact používá se k vkládání studenta a kontaktní osoby zároveň.

tch_UpdateStudent aktualizuje studenta.

tch_UpdateContact aktualizuje kontaktní osobu.

tch_InsertContact vkládá kontakt a uzamyká transakci.

tch_InsertAttendance vkládá docházku a uzamyká transakci.

tch_DeleteAttendance maže docházku a uzamyká transakci.

Příloha B

Obsah přiloženého CD

Na přiloženém CD jsou dvě složky a dva soubory. Soubor obsahující návod k použití CD a informace o spuštěném testovacím systému s názvem readme.txt. A soubor s názvem bakalarska_prace.pdf obsahuje text této bakalářské práce v elektronické podobě. V první ze složek s názvem *databaze*, se nacházejí očíslované skripty pro vytvoření a následné mazání databázových struktur (je třeba je spouštět v pořadí číslování). Tato složka obsahuje i soubor s testovacími daty. V druhé složce jsou s názvem *UI* vloženy soubory pro uživatelské rozhraní.